

SS 2023

Efficient Algorithms and Data Structures II

Harald Räcke

Fakultät für Informatik
TU München

<https://www.moodle.tum.de/course/view.php?id=86234>

Summer Term 2023

Part I

Organizational Matters

Part I

Organizational Matters

- ▶ Modul: IN2004
 - ▶ Name: "Efficient Algorithms and Data Structures II"
"Effiziente Algorithmen und Datenstrukturen II"
 - ▶ ECTS: 8 Credit points
 - ▶ Lectures:
 - ▶ 4 SWS
 - Wed 10:15–11:45 (Room 00.13.009A)
 - Fri 10:15–11:45 (MS HS3)
 - ▶ Webpage:
<https://www.moodle.tum.de/course/view.php?id=86234>

Part I

Organizational Matters

- ▶ Modul: IN2004
- ▶ Name: “Efficient Algorithms and Data Structures II”
“Effiziente Algorithmen und Datenstrukturen II”
- ▶ ECTS: 8 Credit points
- ▶ Lectures:
 - ▶ 4 SWS
Wed 10:15–11:45 (Room 00.13.009A)
Fri 10:15–11:45 (MS HS3)
- ▶ Webpage:
<https://www.moodle.tum.de/course/view.php?id=86234>

Part I

Organizational Matters

- ▶ Modul: IN2004
- ▶ Name: “Efficient Algorithms and Data Structures II”
“Effiziente Algorithmen und Datenstrukturen II”
- ▶ ECTS: 8 Credit points
- ▶ Lectures:
 - ▶ 4 SWS
Wed 10:15–11:45 (Room 00.13.009A)
Fri 10:15–11:45 (MS HS3)
- ▶ Webpage:
<https://www.moodle.tum.de/course/view.php?id=86234>

Part I

Organizational Matters

- ▶ Modul: IN2004
- ▶ Name: “Efficient Algorithms and Data Structures II”
“Effiziente Algorithmen und Datenstrukturen II”
- ▶ ECTS: 8 Credit points
- ▶ Lectures:
 - ▶ 4 SWS
Wed 10:15–11:45 (Room 00.13.009A)
Fri 10:15–11:45 (MS HS3)
- ▶ Webpage:
<https://www.moodle.tum.de/course/view.php?id=86234>

Part I

Organizational Matters

- ▶ Modul: IN2004
- ▶ Name: “Efficient Algorithms and Data Structures II”
“Effiziente Algorithmen und Datenstrukturen II”
- ▶ ECTS: 8 Credit points
- ▶ Lectures:
 - ▶ 4 SWS
Wed 10:15–11:45 (Room 00.13.009A)
Fri 10:15–11:45 (MS HS3)
- ▶ Webpage:
<https://www.moodle.tum.de/course/view.php?id=86234>

The Lecturer

- ▶ Harald Räcke
- ▶ Email: raecke@in.tum.de
- ▶ Room: 03.09.044
- ▶ Office hours: (per appointment)

- ▶ Tutor:
 - ▶ Omar AbdelWanis
 - ▶ omar.abdelwanis@tum.de
 - ▶ per appointment
- ▶ Room: 03.11.018
- ▶ Time: Mon 14:00–16:00

Assessment

- ▶ In order to pass the module you need to pass an exam.

- ▶ Exam:

2.5 hours

There are no resources allowed, except for a calculator and a piece of paper (A4).

Answers should be given in English, but English words accepted.

Assessment

- ▶ In order to pass the module you need to pass an exam.
- ▶ Exam:
 - ▶ 2.5 hours
 - ▶ There are no resources allowed, apart from a hand-written piece of paper (A4).
 - ▶ Answers should be given in English, but German is also accepted.

Assessment

- ▶ In order to pass the module you need to pass an exam.
- ▶ Exam:
 - ▶ 2.5 hours
 - ▶ There are no resources allowed, apart from a hand-written piece of paper (A4).
 - ▶ Answers should be given in English, but German is also accepted.

Assessment

- ▶ In order to pass the module you need to pass an exam.
- ▶ Exam:
 - ▶ 2.5 hours
 - ▶ There are no resources allowed, apart from a hand-written piece of paper (A4).
 - ▶ Answers should be given in English, but German is also accepted.

Assessment

- ▶ In order to pass the module you need to pass an exam.
- ▶ Exam:
 - ▶ 2.5 hours
 - ▶ There are no resources allowed, apart from a hand-written piece of paper (A4).
 - ▶ Answers should be given in English, but German is also accepted.

- ▶ **Assignment Sheets:**

- ▶ An assignment sheet is usually made available on Monday on the module webpage.
- ▶ The first one will be out on Monday, 24 April.

- ▶ Assignment Sheets:
 - ▶ An assignment sheet is usually made available on Monday on the module webpage.
 - ▶ The first one will be out on Monday, 24 April.

- ▶ Assignment Sheets:
 - ▶ An assignment sheet is usually made available on Monday on the module webpage.
 - ▶ The first one will be out on Monday, 24 April.

Part 1: Linear Programming

Part 2: Approximation Algorithms

2 Literatur



V. Chvatal:

Linear Programming,

Freeman, 1983



R. Seidel:

Skript Optimierung, 1996



D. Bertsimas and J.N. Tsitsiklis:

Introduction to Linear Optimization,

Athena Scientific, 1997



Vijay V. Vazirani:

Approximation Algorithms,

Springer 2001



David P. Williamson and David B. Shmoys:
The Design of Approximation Algorithms,
Cambridge University Press 2011



G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A.
Marchetti-Spaccamela, and M. Protasi:
Complexity and Approximation,
Springer, 1999

Part II

Linear Programming

Brewery Problem

Brewery brews ale and beer.

- ▶ Production limited by supply of corn, hops and barley malt
- ▶ Recipes for ale and beer require different amounts of resources

	<i>Corn (kg)</i>	<i>Hops (kg)</i>	<i>Malt (kg)</i>	<i>Profit (€)</i>
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

Brewery Problem

Brewery brews ale and beer.

- ▶ Production limited by supply of corn, hops and barley malt
- ▶ Recipes for ale and beer require different amounts of resources

	<i>Corn (kg)</i>	<i>Hops (kg)</i>	<i>Malt (kg)</i>	<i>Profit (€)</i>
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

Brewery Problem

Brewery brews ale and beer.

- ▶ Production limited by supply of corn, hops and barley malt
- ▶ Recipes for ale and beer require different amounts of resources

	<i>Corn (kg)</i>	<i>Hops (kg)</i>	<i>Malt (kg)</i>	<i>Profit (€)</i>
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

Brewery Problem

	<i>Corn</i> <i>(kg)</i>	<i>Hops</i> <i>(kg)</i>	<i>Malt</i> <i>(kg)</i>	<i>Profit</i> <i>(€)</i>
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow 442 \text{ €}$
- ▶ only brew beer: 32 barrels of beer $\Rightarrow 736 \text{ €}$
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow 775 \text{ €}$
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow 800 \text{ €}$

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 775 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 775 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 775 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 776 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 776 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 776 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

	<i>Corn</i> (kg)	<i>Hops</i> (kg)	<i>Malt</i> (kg)	<i>Profit</i> (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- ▶ only brew ale: 34 barrels of ale $\Rightarrow$ 442 €
- ▶ only brew beer: 32 barrels of beer $\Rightarrow$ 736 €
- ▶ 7.5 barrels ale, 29.5 barrels beer $\Rightarrow$ 776 €
- ▶ 12 barrels ale, 28 barrels beer $\Rightarrow$ 800 €

Brewery Problem

Linear Program

- Introduce variables a and b that define how much ale and stout to produce.
- Choose the variables in such a way that the profit is maximized.
- Make sure that no constraints (due to limited supply) are violated.

$$\begin{array}{ll}\max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0\end{array}$$

Brewery Problem

Linear Program

- ▶ Introduce variables a and b that define how much ale and beer to produce.
- ▶ Choose the variables in such a way that the objective function (profit) is maximized.
- ▶ Make sure that no constraints (due to limited supply) are violated.

$$\begin{array}{llll} \max & 13a & + & 23b \\ \text{s.t.} & 5a & + & 15b \leq 480 \\ & 4a & + & 4b \leq 160 \\ & 35a & + & 20b \leq 1190 \\ & & & a, b \geq 0 \end{array}$$

Brewery Problem

Linear Program

- ▶ Introduce variables a and b that define how much ale and beer to produce.
- ▶ Choose the variables in such a way that the objective function (profit) is maximized.
- ▶ Make sure that no constraints (due to limited supply) are violated.

$$\begin{array}{ll} \max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0 \end{array}$$

Brewery Problem

Linear Program

- ▶ Introduce variables a and b that define how much ale and beer to produce.
- ▶ Choose the variables in such a way that the objective function (profit) is maximized.
- ▶ Make sure that no constraints (due to limited supply) are violated.

$$\begin{array}{ll} \max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0 \end{array}$$

Brewery Problem

Linear Program

- ▶ Introduce variables a and b that define how much ale and beer to produce.
- ▶ Choose the variables in such a way that the objective function (profit) is maximized.
- ▶ Make sure that no constraints (due to limited supply) are violated.

$$\begin{array}{llll} \max & 13a & + & 23b \\ \text{s.t.} & 5a & + & 15b \leq 480 \\ & 4a & + & 4b \leq 160 \\ & 35a & + & 20b \leq 1190 \\ & & & a, b \geq 0 \end{array}$$

Standard Form LPs

LP in standard form:

Maximize $c^T x$
subject to:
Linear constraints: $Ax = b$
nonnegative linear objective function subject to linear inequalities

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \quad 1 \leq i \leq m \\ & x_j \geq 0, \quad 1 \leq j \leq n \end{aligned}$$

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & A_1 x \leq b_1 \\ & A_2 x = b_2 \\ & x_j \geq 0 \end{aligned}$$

Standard Form LPs

LP in standard form:

- ▶ input: numbers a_{ij} , c_j , b_i
- ▶ output: numbers x_j
- ▶ n = #decision variables, m = #constraints
- ▶ maximize linear objective function subject to linear (in)equalities

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j = b_i \quad 1 \leq i \leq m \\ & x_j \geq 0 \quad 1 \leq j \leq n \end{aligned}$$

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Standard Form LPs

LP in standard form:

- ▶ input: numbers a_{ij} , c_j , b_i
- ▶ output: numbers x_j
- ▶ n = #decision variables, m = #constraints
- ▶ maximize linear objective function subject to linear (in)equalities

$$\begin{aligned} \max \quad & c_1 x_1 + c_2 x_2 + \dots + c_n x_n \\ \text{s.t.} \quad & a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n = b_1 \\ & a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n = b_2 \\ & \vdots \\ & a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n = b_m \\ & x_1, x_2, \dots, x_n \geq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & A x = b \\ & x \geq 0 \end{aligned}$$

Standard Form LPs

LP in standard form:

- ▶ input: numbers a_{ij} , c_j , b_i
- ▶ output: numbers x_j
- ▶ n = #decision variables, m = #constraints
- ▶ maximize linear objective function subject to linear (in)equalities

$$\begin{aligned} \max \quad & c_1 x_1 + c_2 x_2 + \dots + c_n x_n \\ \text{s.t.} \quad & a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n = b_1 \\ & a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n = b_2 \\ & \vdots \\ & a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n = b_m \\ & x_1, x_2, \dots, x_n \geq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & c_1 x_1 + c_2 x_2 + \dots + c_n x_n \\ \text{s.t.} \quad & a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n \leq b_1 \\ & a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n \leq b_2 \\ & \vdots \\ & a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n \leq b_m \\ & x_1, x_2, \dots, x_n \geq 0 \end{aligned}$$

Standard Form LPs

LP in standard form:

- ▶ input: numbers a_{ij} , c_j , b_i
- ▶ output: numbers x_j
- ▶ n = #decision variables, m = #constraints
- ▶ maximize linear objective function subject to linear (in)equalities

$$\begin{array}{ll}\max & \sum_{j=1}^n c_j x_j \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j = b_i \quad 1 \leq i \leq m \\ & x_j \geq 0 \quad 1 \leq j \leq n\end{array}$$

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

Standard Form LPs

LP in standard form:

- ▶ input: numbers a_{ij} , c_j , b_i
- ▶ output: numbers x_j
- ▶ n = #decision variables, m = #constraints
- ▶ maximize linear objective function subject to linear (in)equalities

$$\begin{array}{ll}\max & \sum_{j=1}^n c_j x_j \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j = b_i \quad 1 \leq i \leq m \\ & x_j \geq 0 \quad 1 \leq j \leq n\end{array}$$

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

Standard Form LPs

LP in standard form:

- ▶ input: numbers a_{ij} , c_j , b_i
- ▶ output: numbers x_j
- ▶ n = #decision variables, m = #constraints
- ▶ maximize linear objective function subject to linear (in)equalities

$$\begin{array}{ll}\max & \sum_{j=1}^n c_j x_j \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j = b_i \quad 1 \leq i \leq m \\ & x_j \geq 0 \quad 1 \leq j \leq n\end{array}$$

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

Standard Form LPs

Original LP

$$\begin{array}{ll}\max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0\end{array}$$

Standard Form

Add a **slack variable** to every constraint.

$$\begin{array}{llllllll}\max & 13a & + & 23b & & & & \\ \text{s.t.} & 5a & + & 15b & + & s_c & & = 480 \\ & 4a & + & 4b & & & + & s_h = 160 \\ & 35a & + & 20b & & & + & s_m = 1190 \\ & a & , & b & , & s_c & , & s_h , s_m \geq 0\end{array}$$

Standard Form LPs

Original LP

$$\begin{array}{ll}\max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0\end{array}$$

Standard Form

Add a **slack variable** to every constraint.

$$\begin{array}{llllllll}\max & 13a & + & 23b & & & & \\ \text{s.t.} & 5a & + & 15b & + & s_c & & = 480 \\ & 4a & + & 4b & & & + & s_h = 160 \\ & 35a & + & 20b & & & + & s_m = 1190 \\ & a & , & b & , & s_c & , & s_h , s_m \geq 0\end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

standard
maximization form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0\end{array}$$

standard
minimization form

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0\end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

standard
maximization form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0\end{array}$$

standard
minimization form

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0\end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

standard
maximization form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0\end{array}$$

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

standard
minimization form

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0\end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0\end{array}$$

standard
maximization form

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0\end{array}$$

standard
minimization form

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0\end{array}$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

1. Maximize z

2. Minimize z

3. No objective function

4. No constraints

5. Equality constraints

6. Inequality constraints

7. Unbounded feasible region

8. No feasible region

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **less or equal to equality:**

$$a - 3b + 5c \leq 12 \Rightarrow \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- ▶ greater or equal to equality:

$$\begin{aligned} a - 3b + 5c - s &= 12 \\ s &\geq 0 \end{aligned}$$

- ▶ min to max:

$$\min z \Rightarrow \max -z$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **less or equal to equality:**

$$a - 3b + 5c \leq 12 \Rightarrow \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- ▶ greater or equal to equality:

- ▶ min to max:

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **less or equal to equality:**

$$a - 3b + 5c \leq 12 \Rightarrow \begin{array}{l} a - 3b + 5c + s = 12 \\ s \geq 0 \end{array}$$

- ▶ **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \Rightarrow \begin{array}{l} a - 3b + 5c - s = 12 \\ s \geq 0 \end{array}$$

- ▶ **min to max:**

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

► **less or equal to equality:**

$$a - 3b + 5c \leq 12 \Rightarrow \begin{array}{l} a - 3b + 5c + s = 12 \\ s \geq 0 \end{array}$$

► **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \Rightarrow \begin{array}{l} a - 3b + 5c - s = 12 \\ s \geq 0 \end{array}$$

► **min to max:**

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **less or equal to equality:**

$$a - 3b + 5c \leq 12 \Rightarrow \begin{array}{l} a - 3b + 5c + s = 12 \\ s \geq 0 \end{array}$$

- ▶ **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \Rightarrow \begin{array}{l} a - 3b + 5c - s = 12 \\ s \geq 0 \end{array}$$

- ▶ **min to max:**

$$\min a - 3b + 5c \Rightarrow \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **less or equal to equality:**

$$a - 3b + 5c \leq 12 \Rightarrow \begin{array}{l} a - 3b + 5c + s = 12 \\ s \geq 0 \end{array}$$

- ▶ **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \Rightarrow \begin{array}{l} a - 3b + 5c - s = 12 \\ s \geq 0 \end{array}$$

- ▶ **min to max:**

$$\min a - 3b + 5c \Rightarrow \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **equality to less or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{array}{l} a - 3b + 5c \leq 12 \\ -a + 3b - 5c \leq -12 \end{array}$$

- ▶ equality to greater or equal:

- ▶ unrestricted to nonnegative:

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- ▶ **equality to less or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{array}{l} a - 3b + 5c \leq 12 \\ -a + 3b - 5c \leq -12 \end{array}$$

- ▶ equality to greater or equal:

- ▶ unrestricted to nonnegative:

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

► **equality to less or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\leq 12 \\ -a + 3b - 5c &\leq -12 \end{aligned}$$

► **equality to greater or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\geq 12 \\ -a + 3b - 5c &\geq -12 \end{aligned}$$

► **unrestricted to nonnegative:**

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

► **equality to less or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\leq 12 \\ -a + 3b - 5c &\leq -12 \end{aligned}$$

► **equality to greater or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\geq 12 \\ -a + 3b - 5c &\geq -12 \end{aligned}$$

► **unrestricted to nonnegative:**

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

► **equality to less or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\leq 12 \\ -a + 3b - 5c &\leq -12 \end{aligned}$$

► **equality to greater or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\geq 12 \\ -a + 3b - 5c &\geq -12 \end{aligned}$$

► **unrestricted to nonnegative:**

$$x \text{ unrestricted} \Rightarrow x = x^+ - x^-, x^+ \geq 0, x^- \geq 0$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

► **equality to less or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\leq 12 \\ -a + 3b - 5c &\leq -12 \end{aligned}$$

► **equality to greater or equal:**

$$a - 3b + 5c = 12 \Rightarrow \begin{aligned} a - 3b + 5c &\geq 12 \\ -a + 3b - 5c &\geq -12 \end{aligned}$$

► **unrestricted to nonnegative:**

$$x \text{ unrestricted} \Rightarrow x = x^+ - x^-, x^+ \geq 0, x^- \geq 0$$

Observations:

- ▶ a linear program does not contain x^2 , $\cos(x)$, etc.
- ▶ transformations between standard forms can be done efficiently and only change the size of the LP by a small constant factor
- ▶ for the standard minimization or maximization LPs we could include the nonnegativity constraints into the set of ordinary constraints; this is of course not possible for the standard form

Observations:

- ▶ a linear program does not contain x^2 , $\cos(x)$, etc.
- ▶ transformations between standard forms can be done efficiently and only change the size of the LP by a small constant factor
- ▶ for the standard minimization or maximization LPs we could include the nonnegativity constraints into the set of ordinary constraints; this is of course not possible for the standard form

Observations:

- ▶ a linear program does not contain x^2 , $\cos(x)$, etc.
- ▶ transformations between standard forms can be done efficiently and only change the size of the LP by a small constant factor
- ▶ for the standard minimization or maximization LPs we could include the nonnegativity constraints into the set of ordinary constraints; this is of course not possible for the standard form

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

1. Is LP in NP?

2. Is LP in co-NP?

3. Is LP in P?

Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

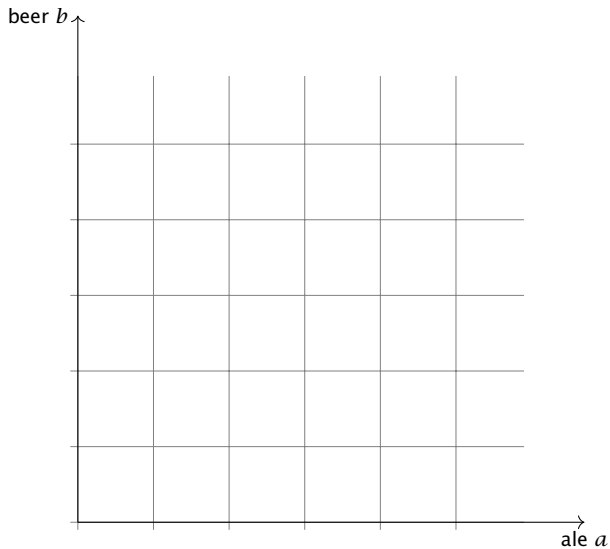
Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

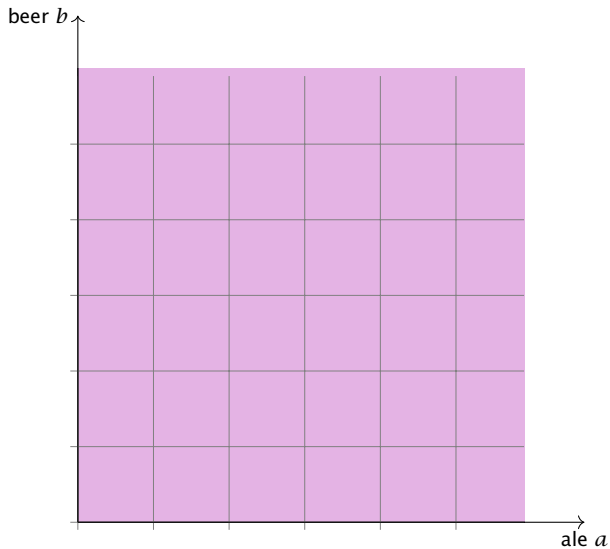
Input size:

- ▶ n number of variables, m constraints, L number of bits to encode the input

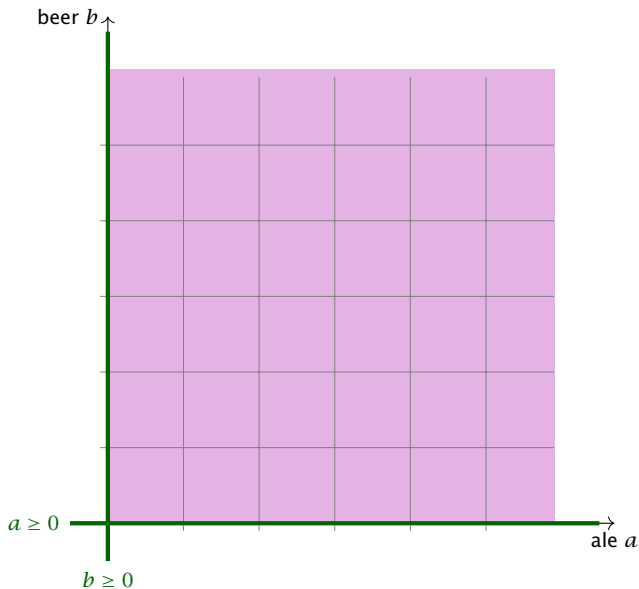
Geometry of Linear Programming



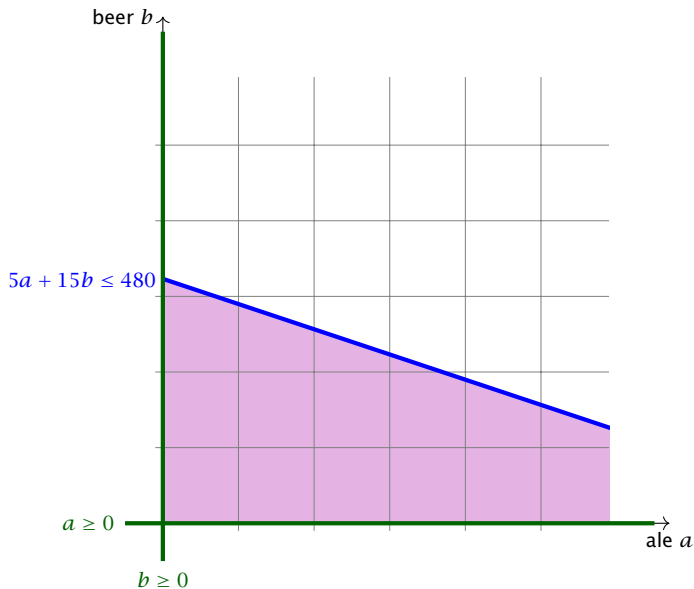
Geometry of Linear Programming



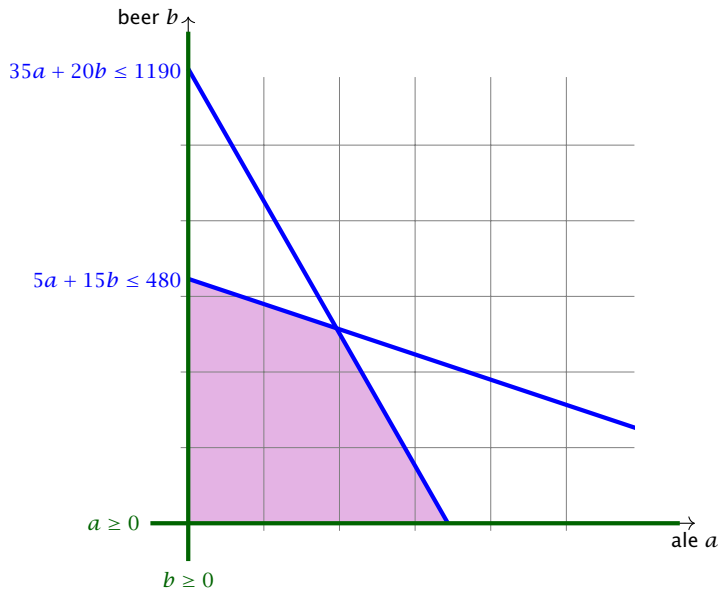
Geometry of Linear Programming



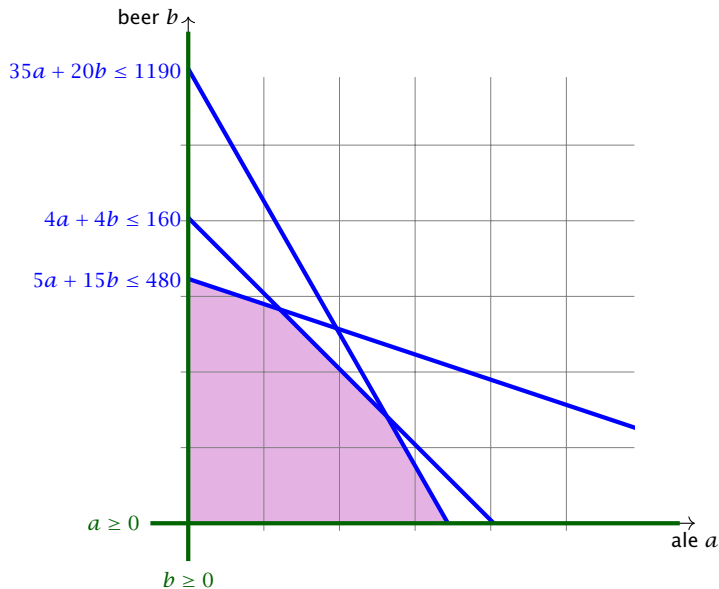
Geometry of Linear Programming



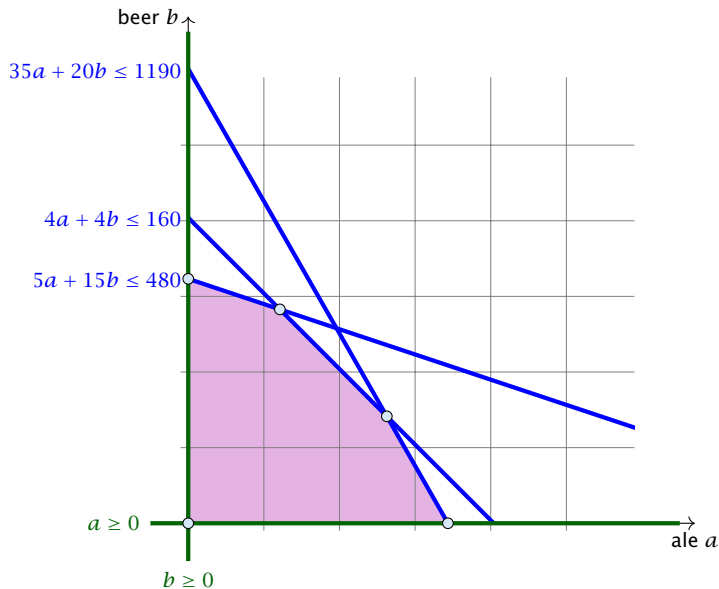
Geometry of Linear Programming



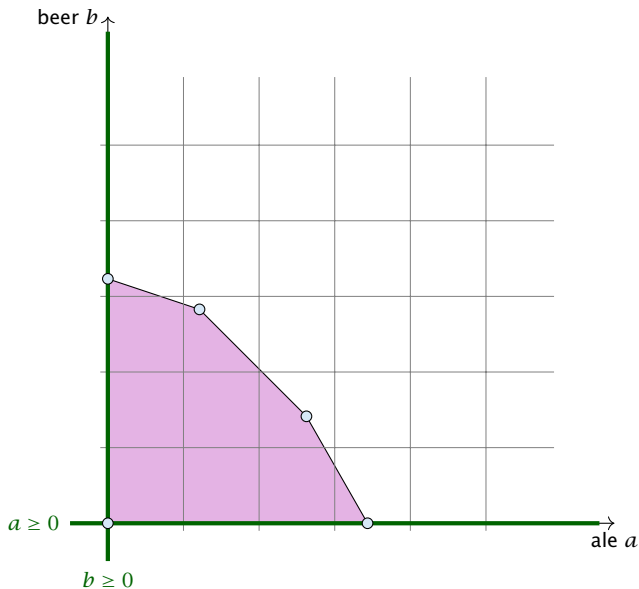
Geometry of Linear Programming



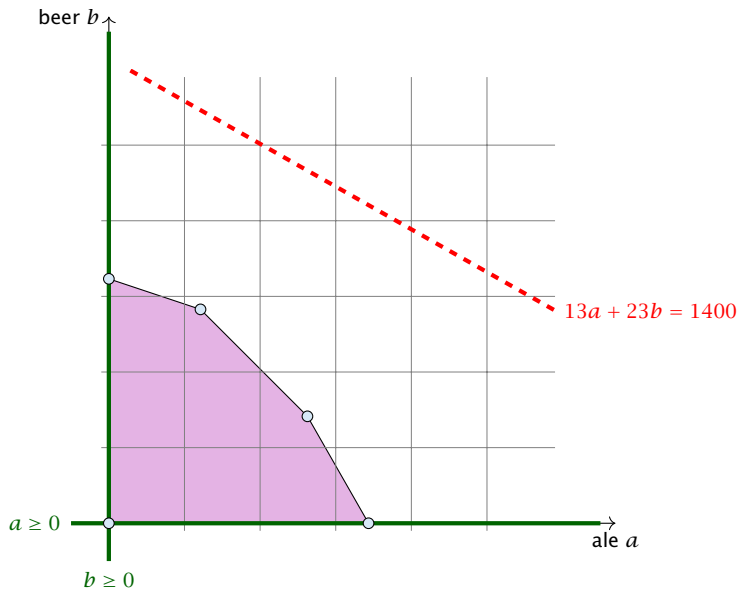
Geometry of Linear Programming



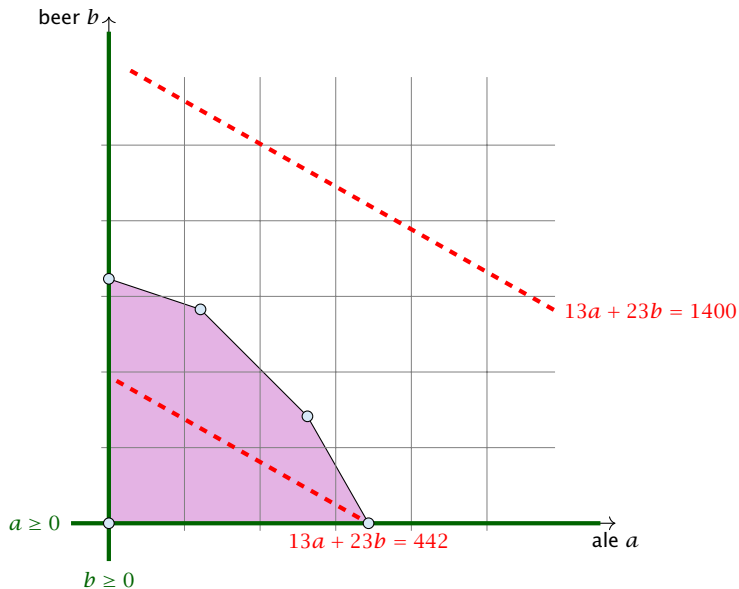
Geometry of Linear Programming



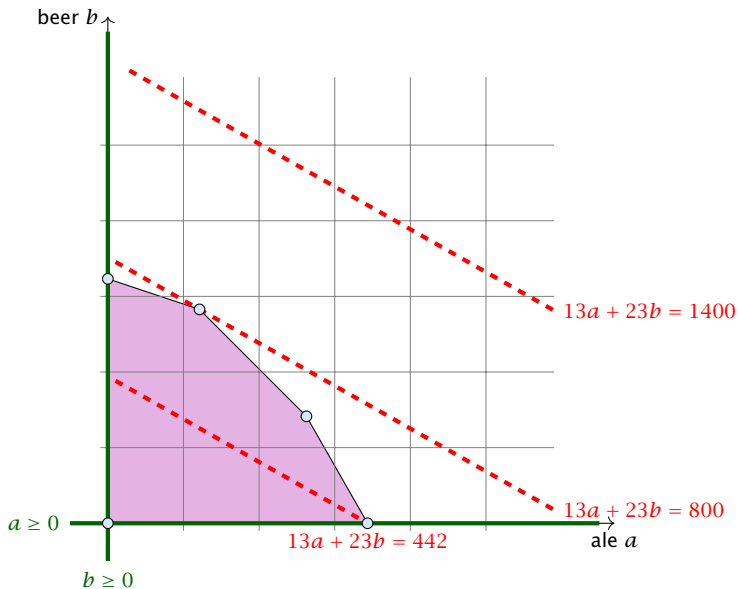
Geometry of Linear Programming



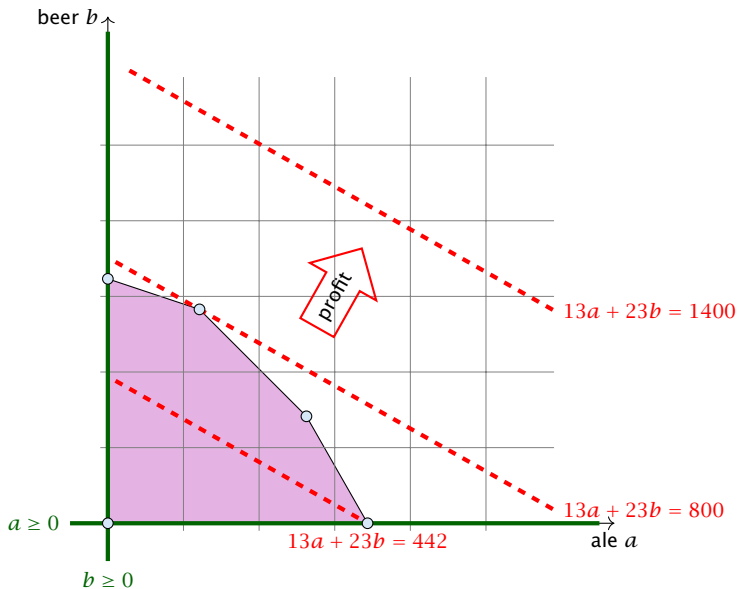
Geometry of Linear Programming



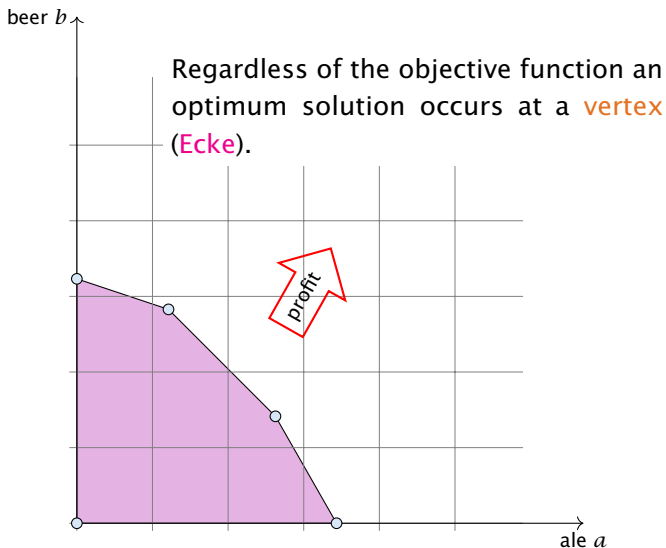
Geometry of Linear Programming



Geometry of Linear Programming



Geometry of Linear Programming



Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (**erfüllbar**). Otherwise, it is called **infeasible** (**unerfüllbar**).
- ▶ An LP is **bounded** (**beschränkt**) if it is feasible and
 – $\exists z^* \in \mathbb{R}$ s.t. $z^* \leq z(x)$ for all $x \in P$ (for maximization problems)
 – $\exists z^* \in \mathbb{R}$ s.t. $z^* \geq z(x)$ for all $x \in P$ (for minimization problems).

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (erfüllbar). Otherwise, it is called **infeasible** (unerfüllbar).
- ▶ An LP is **bounded** (beschränkt) if it is feasible and
if there exists $\alpha \in \mathbb{R}$ for all $x \in P$ the maximization problem
 $\max\{c^T x \mid x \in P\}$ has the value α (for all $x \in P$ the minimization problem
 $\min\{c^T x \mid x \in P\}$ has the value α).

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (**erfüllbar**). Otherwise, it is called **infeasible** (**unerfüllbar**).
- ▶ An LP is **bounded** (**beschränkt**) if it is feasible and
for all $c \in \mathbb{R}^n$ the maximization problem
for all $c \in \mathbb{R}^n$ the minimization problem

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (**erfüllbar**). Otherwise, it is called **infeasible** (**unerfüllbar**).
- ▶ An LP is **bounded** (**beschränkt**) if it is feasible and
for all $c \in \mathbb{R}^n$ the maximization problem
for all $c \in \mathbb{R}^n$ the minimization problem

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (**erfüllbar**). Otherwise, it is called **infeasible** (**unerfüllbar**).
- ▶ An LP is **bounded** (**beschränkt**) if it is feasible and
 - ▶ $c^T x < \infty$ for all $x \in P$ (for maximization problems)
 - ▶ $c^T x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (**erfüllbar**). Otherwise, it is called **infeasible** (**unerfüllbar**).
- ▶ An LP is **bounded** (**beschränkt**) if it is feasible and
 - ▶ $c^T x < \infty$ for all $x \in P$ (for maximization problems)
 - ▶ $c^T x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form

$$P = \{x \mid Ax = b, x \geq 0\}.$$

- ▶ P is called the **feasible region** (**Lösungsraum**) of the LP.
- ▶ A point $x \in P$ is called a **feasible point** (**gültige Lösung**).
- ▶ If $P \neq \emptyset$ then the LP is called **feasible** (**erfüllbar**). Otherwise, it is called **infeasible** (**unerfüllbar**).
- ▶ An LP is **bounded** (**beschränkt**) if it is feasible and
 - ▶ $c^T x < \infty$ for all $x \in P$ (for maximization problems)
 - ▶ $c^T x > -\infty$ for all $x \in P$ (for minimization problems)

Definition 2

Given vectors/points $x_1, \dots, x_k \in \mathbb{R}^n$, $\sum \lambda_i x_i$ is called

- ▶ **linear combination** if $\lambda_i \in \mathbb{R}$.
- ▶ **affine combination** if $\lambda_i \in \mathbb{R}$ and $\sum_i \lambda_i = 1$.
- ▶ **convex combination** if $\lambda_i \in \mathbb{R}$ and $\sum_i \lambda_i = 1$ and $\lambda_i \geq 0$.
- ▶ **conic combination** if $\lambda_i \in \mathbb{R}$ and $\lambda_i \geq 0$.

Note that a combination involves only finitely many vectors.

Definition 3

A set $X \subseteq \mathbb{R}^n$ is called

- ▶ a **linear subspace** if it is closed under linear combinations.
- ▶ an **affine subspace** if it is closed under affine combinations.
- ▶ **convex** if it is closed under convex combinations.
- ▶ a **convex cone** if it is closed under conic combinations.

Note that an affine subspace is **not** a vector space

Definition 4

Given a set $X \subseteq \mathbb{R}^n$.

- ▶ $\text{span}(X)$ is the set of all linear combinations of X
(linear hull, span)
- ▶ $\text{aff}(X)$ is the set of all affine combinations of X
(affine hull)
- ▶ $\text{conv}(X)$ is the set of all convex combinations of X
(convex hull)
- ▶ $\text{cone}(X)$ is the set of all conic combinations of X
(conic hull)

Definition 5

A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is **convex** if for $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$ we have

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

Lemma 6

If $P \subseteq \mathbb{R}^n$, and $f: \mathbb{R}^n \rightarrow \mathbb{R}$ convex then also

$$Q = \{x \in P \mid f(x) \leq t\}$$

Definition 5

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is **convex** if for $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$ we have

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

Lemma 6

If $P \subseteq \mathbb{R}^n$, and $f : \mathbb{R}^n \rightarrow \mathbb{R}$ convex then also

$$Q = \{x \in P \mid f(x) \leq t\}$$

Dimensions

Definition 7

The **dimension** $\dim(A)$ of an affine subspace $A \subseteq \mathbb{R}^n$ is the dimension of the vector space $\{x - a \mid x \in A\}$, where $a \in A$.

Definition 8

The **dimension** $\dim(X)$ of a convex set $X \subseteq \mathbb{R}^n$ is the dimension of its affine hull $\text{aff}(X)$.

Definition 9

A set $H \subseteq \mathbb{R}^n$ is a **hyperplane** if $H = \{x \mid a^T x = b\}$, for $a \neq 0$.

Definition 10

A set $H' \subseteq \mathbb{R}^n$ is a (closed) **halfspace** if $H = \{x \mid a^T x \leq b\}$, for $a \neq 0$.

Definition 9

A set $H \subseteq \mathbb{R}^n$ is a **hyperplane** if $H = \{x \mid a^T x = b\}$, for $a \neq 0$.

Definition 10

A set $H' \subseteq \mathbb{R}^n$ is a (closed) **halfspace** if $H = \{x \mid a^T x \leq b\}$, for $a \neq 0$.

Definitions

Definition 11

A **polytop** is a set $P \subseteq \mathbb{R}^n$ that is the convex hull of a **finite** set of points, i.e., $P = \text{conv}(X)$ where $|X| = c$.

Definitions

Definition 12

A **polyhedron** is a set $P \subseteq \mathbb{R}^n$ that can be represented as the intersection of **finitely** many half-spaces $\{H(a_1, b_1), \dots, H(a_m, b_m)\}$, where

$$H(a_i, b_i) = \{x \in \mathbb{R}^n \mid a_i x \leq b_i\} \quad .$$

Definition 13

A polyhedron P is **bounded** if there exists B s.t. $\|x\|_2 \leq B$ for all $x \in P$.

Definitions

Definition 12

A **polyhedron** is a set $P \subseteq \mathbb{R}^n$ that can be represented as the intersection of **finitely** many half-spaces $\{H(a_1, b_1), \dots, H(a_m, b_m)\}$, where

$$H(a_i, b_i) = \{x \in \mathbb{R}^n \mid a_i x \leq b_i\} \quad .$$

Definition 13

A polyhedron P is **bounded** if there exists B s.t. $\|x\|_2 \leq B$ for all $x \in P$.

Theorem 14

P is a bounded polyhedron iff P is a polytop.

Definition 15

Let $P \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane

$$H(a, b) = \{x \in \mathbb{R}^n \mid a^T x = b\}$$

is a **supporting hyperplane** of P if $\max\{a^T x \mid x \in P\} = b$.

Definition 16

Let $P \subseteq \mathbb{R}^n$. F is a **face** of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H .

Definition 17

Let $P \subseteq \mathbb{R}^n$.

- ▶ a face v is a **vertex** of P if $\{v\}$ is a face of P .
- ▶ a face e is an **edge** of P if e is a face and $\dim(e) = 1$.
- ▶ a face F is a **facet** of P if F is a face and $\dim(F) = \dim(P) - 1$.

Definition 15

Let $P \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane

$$H(a, b) = \{x \in \mathbb{R}^n \mid a^T x = b\}$$

is a **supporting hyperplane** of P if $\max\{a^T x \mid x \in P\} = b$.

Definition 16

Let $P \subseteq \mathbb{R}^n$. F is a **face** of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H .

Definition 17

Let $P \subseteq \mathbb{R}^n$.

- ▶ a face v is a **vertex** of P if $\{v\}$ is a face of P .
- ▶ a face e is an **edge** of P if e is a face and $\dim(e) = 1$.
- ▶ a face F is a **facet** of P if F is a face and $\dim(F) = \dim(P) - 1$.

Definition 15

Let $P \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane

$$H(a, b) = \{x \in \mathbb{R}^n \mid a^T x = b\}$$

is a **supporting hyperplane** of P if $\max\{a^T x \mid x \in P\} = b$.

Definition 16

Let $P \subseteq \mathbb{R}^n$. F is a **face** of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H .

Definition 17

Let $P \subseteq \mathbb{R}^n$.

- ▶ a face v is a **vertex** of P if $\{v\}$ is a face of P .
- ▶ a face e is an **edge** of P if e is a face and $\dim(e) = 1$.
- ▶ a face F is a **facet** of P if F is a face and $\dim(F) = \dim(P) - 1$.

Equivalent definition for vertex:

Definition 18

Given polyhedron P . A point $x \in P$ is a **vertex** if $\exists c \in \mathbb{R}^n$ such that $c^T y < c^T x$, for all $y \in P$, $y \neq x$.

Definition 19

Given polyhedron P . A point $x \in P$ is an **extreme point** if $\nexists a, b \neq x$, $a, b \in P$, with $\lambda a + (1 - \lambda)b = x$ for $\lambda \in [0, 1]$.

Lemma 20

A vertex is also an extreme point.

Equivalent definition for vertex:

Definition 18

Given polyhedron P . A point $x \in P$ is a **vertex** if $\exists c \in \mathbb{R}^n$ such that $c^T y < c^T x$, for all $y \in P$, $y \neq x$.

Definition 19

Given polyhedron P . A point $x \in P$ is an **extreme point** if $\nexists a, b \neq x$, $a, b \in P$, with $\lambda a + (1 - \lambda)b = x$ for $\lambda \in [0, 1]$.

Lemma 20

A vertex is also an extreme point.

Observation

The feasible region of an LP is a Polyhedron.

Theorem 21

If there exists an optimal solution to an LP (in standard form) then there exists an optimum solution that is an extreme point.

Proof

- ▶ suppose x is optimal solution that is not extreme point
- ▶ there exists direction $d \neq 0$ such that $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ Wlog. assume $c^T d \geq 0$ (by taking either d or $-d$)
- ▶ Consider $x + \lambda d, \lambda > 0$

Theorem 21

If there exists an optimal solution to an LP (in standard form) then there exists an optimum solution that is an extreme point.

Proof

- ▶ suppose x is optimal solution that is not extreme point
- ▶ there exists direction $d \neq 0$ such that $x \pm d \in P$
 - ▶ $Ad = 0$ because $A(x \pm d) = b$
 - ▶ Wlog. assume $c^T d \geq 0$ (by taking either d or $-d$)
 - ▶ Consider $x + \lambda d, \lambda > 0$

Theorem 21

If there exists an optimal solution to an LP (in standard form) then there exists an optimum solution that is an extreme point.

Proof

- ▶ suppose x is optimal solution that is not extreme point
- ▶ there exists direction $d \neq 0$ such that $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ Wlog. assume $c^T d \geq 0$ (by taking either d or $-d$)
- ▶ Consider $x + \lambda d, \lambda > 0$

Theorem 21

If there exists an optimal solution to an LP (in standard form) then there exists an optimum solution that is an extreme point.

Proof

- ▶ suppose x is optimal solution that is not extreme point
- ▶ there exists direction $d \neq 0$ such that $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ Wlog. assume $c^T d \geq 0$ (by taking either d or $-d$)
- ▶ Consider $x + \lambda d, \lambda > 0$

Theorem 21

If there exists an optimal solution to an LP (in standard form) then there exists an optimum solution that is an extreme point.

Proof

- ▶ suppose x is optimal solution that is not extreme point
- ▶ there exists direction $d \neq 0$ such that $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ Wlog. assume $c^T d \geq 0$ (by taking either d or $-d$)
- ▶ Consider $x + \lambda d, \lambda > 0$

Convex Sets

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

• Increase λ to λ^* until first component of $d + \lambda c$ hits 0.
• $d + \lambda^* c$ is feasible. Since $-c^T d = -c^T(d + \lambda^* c) = 0$ and $d + \lambda^* c \geq 0$,
• $d + \lambda^* c$ has the more zero-component $\lambda_j = 0$ for some j ($d_j < 0$).
• (Another λ gives $d + \lambda c \geq 0$ for $\lambda > \lambda^*$.)

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

• $d + \lambda c$ is feasible for all $\lambda \geq 0$ since $d_j \geq 0$ and $d + \lambda c \geq 0$ and
• $c^T(d + \lambda c) = c^T d + \lambda c^T c > 0$ for all $\lambda \geq 0$.
• as $\lambda \rightarrow \infty$, $d + \lambda c \rightarrow \infty$ as $c^T c > 0$.

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

Consider $y = 0$ until first component of d that is < 0 . This y is feasible. Since $c^T d > 0$ and $d_j < 0$, $c_j > 0$. y has the more zero-component $d_j < 0$ for case 1. (if $d_j = 0$ for all j)
If $d_j < 0$ for all j , then $c_j > 0$ for all j .
If $d_j < 0$ for all j , then $c_j > 0$ for all j .

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

$y = 0$ is feasible for all $\lambda \geq 0$ since $d_j \geq 0$ and $c_j \geq 0$.
If $d_j \geq 0$ for all j and $c^T d > 0$, then $c_j > 0$ for all j .
If $d_j \geq 0$ for all j and $c^T d > 0$, then $c_j > 0$ for all j .

Case 1. [$\exists j$ s.t. $d_j < 0$]

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$.
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. [$d_j \geq 0$ for all j and $c^T d > 0$]

- ▶ $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq 0$.
- ▶ $c^T x + \lambda c^T d > c^T x$ for all $\lambda > 0$ since $c^T d > 0$.
- ▶ P is unbounded and x is not optimal.

Case 1. [$\exists j$ s.t. $d_j < 0$]

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. [$d_j \geq 0$ for all j and $c^T d > 0$]

- ▶ $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $x + \lambda d \geq 0$ and $A(x + \lambda d) = b$
- ▶ $c^T x + \lambda c^T d > c^T x$ for all $\lambda > 0$
- ▶ x is not optimal

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

- ▶ $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $x + \lambda d \geq 0$ and $A(x + \lambda d) = b$
- ▶ $c^T x + \lambda c^T d > c^T x$ for all $\lambda > 0$
- ▶ x is not optimal

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

- ▶ $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- ▶ as $\lambda \rightarrow \infty$, $c^T (x + \lambda d) \rightarrow \infty$ as $c^T d > 0$

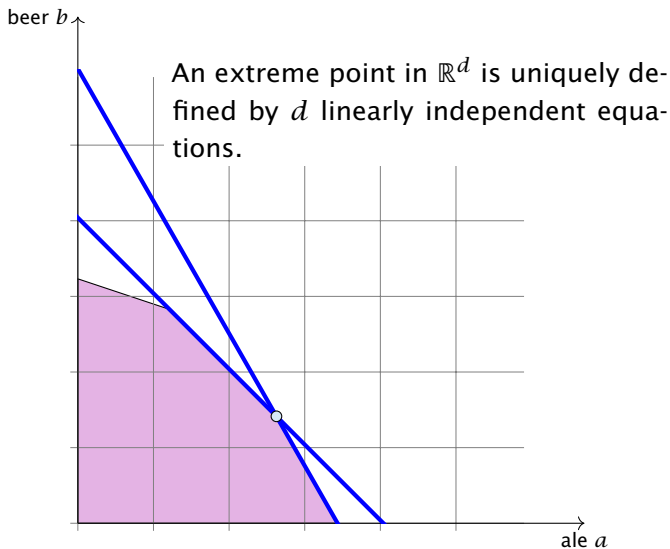
Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- ▶ increase λ to λ' until first component of $x + \lambda d$ hits 0
- ▶ $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- ▶ $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- ▶ $c^T x' = c^T (x + \lambda' d) = c^T x + \lambda' c^T d \geq c^T x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^T d > 0]$

- ▶ $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- ▶ as $\lambda \rightarrow \infty$, $c^T (x + \lambda d) \rightarrow \infty$ as $c^T d > 0$

Algebraic View



Notation

Suppose $B \subseteq \{1 \dots n\}$ is a set of column-indices. Define A_B as the subset of columns of A indexed by B .

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is extreme point iff A_B has linearly independent columns.

Notation

Suppose $B \subseteq \{1 \dots n\}$ is a set of column-indices. Define A_B as the subset of columns of A indexed by B .

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ define $B' = \{j \mid d_j \neq 0\}$
- ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
- ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
 - ▶ $Ad = 0$ because $A(x \pm d) = b$
 - ▶ define $B' = \{j \mid d_j \neq 0\}$
 - ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
 - ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
 - ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ define $B' = \{j \mid d_j \neq 0\}$
- ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
- ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ define $B' = \{j \mid d_j \neq 0\}$
- ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
- ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ define $B' = \{j \mid d_j \neq 0\}$
- ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
 - ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
 - ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ define $B' = \{j \mid d_j \neq 0\}$
- ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
- ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Leftarrow$)

- ▶ assume x is not extreme point
- ▶ there exists direction d s.t. $x \pm d \in P$
- ▶ $Ad = 0$ because $A(x \pm d) = b$
- ▶ define $B' = \{j \mid d_j \neq 0\}$
- ▶ $A_{B'}$ has linearly dependent columns as $Ad = 0$
- ▶ $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- ▶ Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

- ▶ assume A_B has linearly dependent columns
- ▶ there exists $d \neq 0$ such that $A_B d = 0$
- ▶ extend d to $\mathbb{R}^n$ by adding 0-components
- ▶ now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- ▶ for sufficiently small λ we have $x \pm \lambda d \in P$
- ▶ hence, x is not extreme point

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

- ▶ assume A_B has linearly dependent columns
- ▶ there exists $d \neq 0$ such that $A_B d = 0$
 - ▶ extend d to $\mathbb{R}^n$ by adding 0-components
 - ▶ now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
 - ▶ for sufficiently small λ we have $x \pm \lambda d \in P$
 - ▶ hence, x is not extreme point

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

- ▶ assume A_B has linearly dependent columns
- ▶ there exists $d \neq 0$ such that $A_B d = 0$
- ▶ extend d to $\mathbb{R}^n$ by adding 0-components
- ▶ now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- ▶ for sufficiently small λ we have $x \pm \lambda d \in P$
- ▶ hence, x is not extreme point

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

- ▶ assume A_B has linearly dependent columns
- ▶ there exists $d \neq 0$ such that $A_B d = 0$
- ▶ extend d to $\mathbb{R}^n$ by adding 0-components
- ▶ now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- ▶ for sufficiently small λ we have $x \pm \lambda d \in P$
- ▶ hence, x is not extreme point

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

- ▶ assume A_B has linearly dependent columns
- ▶ there exists $d \neq 0$ such that $A_B d = 0$
- ▶ extend d to $\mathbb{R}^n$ by adding 0-components
- ▶ now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- ▶ for sufficiently small λ we have $x \pm \lambda d \in P$
- ▶ hence, x is not extreme point

Theorem 22

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$.
Then x is extreme point **iff** A_B has linearly independent columns.

Proof ($\Rightarrow$)

- ▶ assume A_B has linearly dependent columns
- ▶ there exists $d \neq 0$ such that $A_B d = 0$
- ▶ extend d to $\mathbb{R}^n$ by adding 0-components
- ▶ now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- ▶ for sufficiently small λ we have $x \pm \lambda d \in P$
- ▶ hence, x is not extreme point

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
- ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
- ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
- ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
- ▶ we get $y = x$
- ▶ hence, x is a vertex of P

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
 - ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
 - ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
 - ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
 - ▶ we get $y = x$
 - ▶ hence, x is a vertex of P

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
- ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
- ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
- ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
- ▶ we get $y = x$
- ▶ hence, x is a vertex of P

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
- ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
- ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
 - ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
 - ▶ we get $y = x$
 - ▶ hence, x is a vertex of P

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
- ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
- ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
- ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
- ▶ we get $y = x$
- ▶ hence, x is a vertex of P

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
- ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
- ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
- ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
- ▶ we get $y = x$
- ▶ hence, x is a vertex of P

Theorem 23

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. If A_B has linearly independent columns then x is a vertex of P .

- ▶ define $c_j = \begin{cases} 0 & j \in B \\ -1 & j \notin B \end{cases}$
- ▶ then $c^T x = 0$ and $c^T y \leq 0$ for $y \in P$
- ▶ assume $c^T y = 0$; then $y_j = 0$ for all $j \notin B$
- ▶ $b = Ay = A_B y_B = Ax = A_B x_B$ gives that $A_B(x_B - y_B) = 0$;
- ▶ this means that $x_B = y_B$ since A_B has linearly independent columns
- ▶ we get $y = x$
- ▶ hence, x is a vertex of P

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

C1 if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_2 x \leq b_2, \dots, A_m x \leq b_m$ we also have $A_1 x \leq b_1$, hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_2 x \leq b_2, \dots, A_m x \leq b_m$ we also have $A_1 x \leq b_1$, hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_2 x \leq b_2, \dots, A_m x \leq b_m$ we also have $A_1 x \leq b_1$, hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

- C1** if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous
- C2** if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^m \lambda_i \cdot A_i x = \sum_{i=2}^m \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

- C1** if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous
- C2** if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^m \lambda_i \cdot A_i x = \sum_{i=2}^m \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

- C1** if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous
- C2** if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^m \lambda_i \cdot A_i x = \sum_{i=2}^m \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

- C1** if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous
- C2** if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^m \lambda_i \cdot A_i x = \sum_{i=2}^m \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- ▶ assume that $\text{rank}(A) < m$
- ▶ assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^m \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

- C1** if now $b_1 = \sum_{i=2}^m \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous
- C2** if $b_1 \neq \sum_{i=2}^m \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^m \lambda_i \cdot A_i x = \sum_{i=2}^m \lambda_i \cdot b_i \neq b_1$$

From now on we will always assume that the constraint matrix of a standard form LP has full row rank.

Theorem 24

Given $P = \{x \mid Ax = b, x \geq 0\}$. x is extreme point iff there exists $B \subseteq \{1, \dots, n\}$ with $|B| = m$ and

- ▶ A_B is non-singular
- ▶ $x_B = A_B^{-1}b \geq 0$
- ▶ $x_N = 0$

where $N = \{1, \dots, n\} \setminus B$.

Proof

Take $B = \{j \mid x_j > 0\}$ and augment with linearly independent columns until $|B| = m$; always possible since $\text{rank}(A) = m$.

Theorem 24

Given $P = \{x \mid Ax = b, x \geq 0\}$. x is extreme point iff there exists $B \subseteq \{1, \dots, n\}$ with $|B| = m$ and

- ▶ A_B is non-singular
- ▶ $x_B = A_B^{-1}b \geq 0$
- ▶ $x_N = 0$

where $N = \{1, \dots, n\} \setminus B$.

Proof

Take $B = \{j \mid x_j > 0\}$ and augment with linearly independent columns until $|B| = m$; always possible since $\text{rank}(A) = m$.

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called **basic solution** (Basislösung) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a **basic feasible solution** (gültige Basislösung) if in addition $x \geq 0$.

A **basis** (Basis) is an index set $B \subseteq \{1, \dots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x_B = b$ and $x_j = 0$ for all $j \notin B$ is **the basic solution associated to basis B** (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called **basic solution** (**Basislösung**) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a **basic feasible solution** (gültige Basislösung) if in addition $x \geq 0$.

A **basis** (Basis) is an index set $B \subseteq \{1, \dots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x_B = b$ and $x_j = 0$ for all $j \notin B$ is the **basic solution associated to basis B** (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called **basic solution** (**Basislösung**) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a **basic feasible solution** (**gültige Basislösung**) if in addition $x \geq 0$.

A **basis** (**Basis**) is an index set $B \subseteq \{1, \dots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x_B = b$ and $x_j = 0$ for all $j \notin B$ is the **basic solution associated to basis B** (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called **basic solution** (**Basislösung**) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a **basic feasible solution** (**gültige Basislösung**) if in addition $x \geq 0$.

A **basis** (**Basis**) is an index set $B \subseteq \{1, \dots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x_B = b$ and $x_j = 0$ for all $j \notin B$ is the **basic solution associated to basis B** (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called **basic solution** (**Basislösung**) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a **basic feasible solution** (**gültige Basislösung**) if in addition $x \geq 0$.

A **basis** (**Basis**) is an index set $B \subseteq \{1, \dots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x_B = b$ and $x_j = 0$ for all $j \notin B$ is the **basic solution associated to basis B** (**die zu B assoziierte Basislösung**)

Basic Feasible Solutions

A BFS fulfills the m equality constraints.

In addition, at least $n - m$ of the x_i 's are zero. The corresponding non-negativity constraint is fulfilled with equality.

Fact:

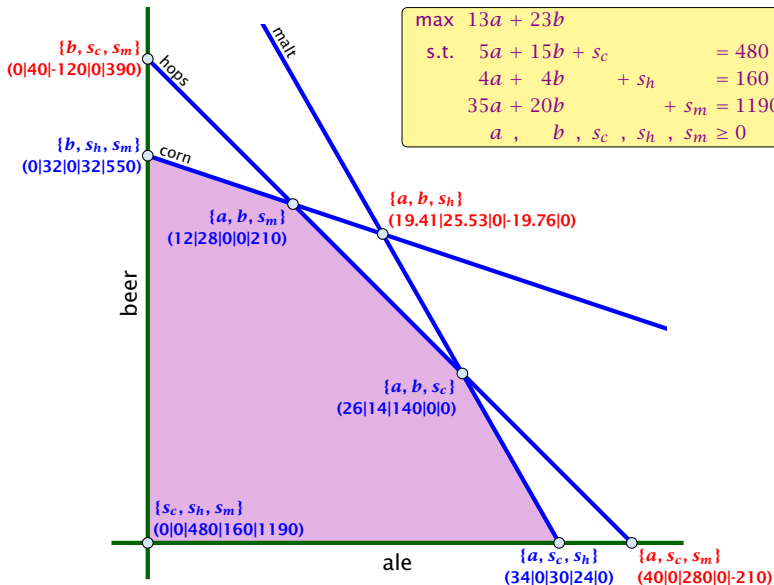
In a BFS at least n constraints are fulfilled with equality.

Basic Feasible Solutions

Definition 25

For a general LP ($\max\{c^T x \mid Ax \leq b\}$) with n variables a point x is a **basic feasible solution** if x is feasible and there exist n (linearly independent) constraints that are tight.

Algebraic View



Fundamental Questions

Linear Programming Problem (LP)

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$
s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP? yes!
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

Proof:

- ▶ Given a basis B we can compute the associated basis solution by calculating $A_B^{-1}b$ in polynomial time; then we can also compute the profit.

Fundamental Questions

Linear Programming Problem (LP)

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP? yes!
- ▶ Is LP in co-NP?
- ▶ Is LP in P?

Proof:

- ▶ Given a basis B we can compute the associated basis solution by calculating $A_B^{-1}b$ in polynomial time; then we can also compute the profit.

Observation

We can compute an optimal solution to a linear program in time $\mathcal{O}\left(\binom{n}{m} \cdot \text{poly}(n, m)\right)$.

- ▶ there are only $\binom{n}{m}$ different bases.
- ▶ compute the profit of each of them and take the maximum

What happens if LP is unbounded?

4 Simplex Algorithm

Enumerating all basic feasible solutions (BFS), in order to find the optimum is slow.

Simplex Algorithm [George Dantzig 1947]

Move from BFS to adjacent BFS, without decreasing objective function.

Two BFSs are called adjacent if the bases just differ in one variable.

4 Simplex Algorithm

Enumerating all basic feasible solutions (BFS), in order to find the optimum is slow.

Simplex Algorithm [George Dantzig 1947]

Move from BFS to **adjacent** BFS, without decreasing objective function.

Two BFSs are called **adjacent** if the bases just differ in one variable.

4 Simplex Algorithm

$$\begin{array}{llll} \max & 13a + 23b & & \\ \text{s.t.} & 5a + 15b + s_c & & = 480 \\ & 4a + 4b & + s_h & = 160 \\ & 35a + 20b & & + s_m = 1190 \\ & a, b, s_c, s_h, s_m & \geq & 0 \end{array}$$

$$\begin{array}{llll} \max & Z & & \\ & 13a + 23b & & - Z = 0 \\ & 5a + 15b + s_c & & = 480 \\ & 4a + 4b & + s_h & = 160 \\ & 35a + 20b & & + s_m = 1190 \\ & a, b, s_c, s_h, s_m & \geq & 0 \end{array}$$

$$\text{basis} = \{s_c, s_h, s_m\}$$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

4 Simplex Algorithm

$$\begin{array}{llll} \max & 13a + 23b & & \\ \text{s.t.} & 5a + 15b + s_c & = & 480 \\ & 4a + 4b + s_h & = & 160 \\ & 35a + 20b + s_m & = & 1190 \\ & a, b, s_c, s_h, s_m & \geq & 0 \end{array}$$

$$\begin{array}{llll} \max & Z & & \\ & 13a + 23b & - Z = & 0 \\ & 5a + 15b + s_c & = & 480 \\ & 4a + 4b + s_h & = & 160 \\ & 35a + 20b + s_m & = & 1190 \\ & a, b, s_c, s_h, s_m & \geq & 0 \end{array}$$

$$\text{basis} = \{s_c, s_h, s_m\}$$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

Pivoting Step

max Z

$$13a + 23b \quad \quad \quad - Z = 0$$

$$5a + 15b + s_c \quad \quad \quad = 480$$

$$4a + 4b \quad \quad + s_h \quad \quad \quad = 160$$

$$35a + 20b \quad \quad \quad + s_m \quad \quad \quad = 1190$$

$$a, \quad b, \quad s_c, \quad s_h, \quad s_m \quad \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

Pivoting Step

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

- ▶ choose variable to bring into the basis
 - ▶ chosen variable should have positive coefficient in objective function
 - ▶ apply min-ratio test to find out by how much the variable can be increased
 - ▶ pivot on row found by min-ratio test
 - ▶ the existing basis variable in this row leaves the basis

Pivoting Step

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

- ▶ choose variable to bring into the basis
- ▶ chosen variable should have positive coefficient in objective function
- ▶ apply min-ratio test to find out by how much the variable can be increased
- ▶ pivot on row found by min-ratio test
- ▶ the existing basis variable in this row leaves the basis

Pivoting Step

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- ▶ choose variable to bring into the basis
- ▶ chosen variable should have positive coefficient in objective function
- ▶ apply **min-ratio** test to find out by how much the variable can be increased
- ▶ pivot on row found by min-ratio test
- ▶ the existing basis variable in this row leaves the basis

Pivoting Step

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- ▶ choose variable to bring into the basis
- ▶ chosen variable should have positive coefficient in objective function
- ▶ apply **min-ratio** test to find out by how much the variable can be increased
- ▶ pivot on row found by min-ratio test
- ▶ the existing basis variable in this row leaves the basis

Pivoting Step

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- ▶ choose variable to bring into the basis
- ▶ chosen variable should have positive coefficient in objective function
- ▶ apply **min-ratio** test to find out by how much the variable can be increased
- ▶ pivot on row found by min-ratio test
- ▶ the existing basis variable in this row leaves the basis

max Z

$$13a + 23b \quad \quad \quad - Z = 0$$

$$5a + 15b + s_c \quad \quad \quad = 480$$

$$4a + 4b \quad \quad + s_h \quad \quad \quad = 160$$

$$35a + 20b \quad \quad \quad + s_m \quad \quad \quad = 1190$$

$$a, \quad b, \quad s_c, \quad s_h, \quad s_m \quad \quad \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- Choose variable with coefficient > 0 as entering variable.

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

- ▶ Choose variable with coefficient > 0 as entering variable.
- ▶ If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- ▶ Choose variable with coefficient > 0 as entering variable.
- ▶ If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.
- ▶ For maintaining $Ax = b$ we need e.g. to set $s_c = 480 - 15\theta$.

max Z

$$13a + 23b \quad - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- ▶ Choose variable with coefficient > 0 as entering variable.
- ▶ If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.
- ▶ For maintaining $Ax = b$ we need e.g. to set $s_c = 480 - 15\theta$.
- ▶ Choosing $\theta = \min\{480/15, 160/4, 1190/20\}$ ensures that in the new solution one current basic variable becomes 0, and no variable goes negative.

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- ▶ Choose variable with coefficient > 0 as **entering variable**.
- ▶ If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.
- ▶ For maintaining $Ax = b$ we need e.g. to set $s_c = 480 - 15\theta$.
- ▶ Choosing $\theta = \min\{480/15, 160/4, 1190/20\}$ ensures that in the new solution one current basic variable becomes 0, and no variable goes negative.
- ▶ The basic variable in the row that gives $\min\{480/15, 160/4, 1190/20\}$ becomes the **leaving variable**.

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

Substitute $b = \frac{1}{15}(480 - 5a - s_c)$.

max Z

$$13a + 23b - Z = 0$$

$$5a + 15b + s_c = 480$$

$$4a + 4b + s_h = 160$$

$$35a + 20b + s_m = 1190$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{s_c, s_h, s_m\}$

$$a = b = 0$$

$$Z = 0$$

$$s_c = 480$$

$$s_h = 160$$

$$s_m = 1190$$

Substitute $b = \frac{1}{15}(480 - 5a - s_c)$.

max Z

$$\frac{16}{3}a - \frac{23}{15}s_c - Z = -736$$

$$\frac{1}{3}a + b + \frac{1}{15}s_c = 32$$

$$\frac{8}{3}a - \frac{4}{15}s_c + s_h = 32$$

$$\frac{85}{3}a - \frac{4}{3}s_c + s_m = 550$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{b, s_h, s_m\}$

$$a = s_c = 0$$

$$Z = 736$$

$$b = 32$$

$$s_h = 32$$

$$s_m = 550$$

max Z

$$\frac{16}{3}a - \frac{23}{15}s_c - Z = -736$$

$$\frac{1}{3}a + b + \frac{1}{15}s_c = 32$$

$$\frac{8}{3}a - \frac{4}{15}s_c + s_h = 32$$

$$\frac{85}{3}a - \frac{4}{3}s_c + s_m = 550$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{b, s_h, s_m\}$

$$a = s_c = 0$$

$$Z = 736$$

$$b = 32$$

$$s_h = 32$$

$$s_m = 550$$

max Z

$$\frac{16}{3}a - \frac{23}{15}s_c - Z = -736$$

$$\frac{1}{3}a + b + \frac{1}{15}s_c = 32$$

$$\frac{8}{3}a - \frac{4}{15}s_c + s_h = 32$$

$$\frac{85}{3}a - \frac{4}{3}s_c + s_m = 550$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{b, s_h, s_m\}$

$$a = s_c = 0$$

$$Z = 736$$

$$b = 32$$

$$s_h = 32$$

$$s_m = 550$$

Choose variable a to bring into basis.

max Z

$$\frac{16}{3}a - \frac{23}{15}s_c - Z = -736$$

$$\frac{1}{3}a + b + \frac{1}{15}s_c = 32$$

$$\frac{8}{3}a - \frac{4}{15}s_c + s_h = 32$$

$$\frac{85}{3}a - \frac{4}{3}s_c + s_m = 550$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{b, s_h, s_m\}$

$a = s_c = 0$

$Z = 736$

$b = 32$

$s_h = 32$

$s_m = 550$

Choose variable a to bring into basis.

Computing $\min\{3 \cdot 32, 3 \cdot 32/8, 3 \cdot 550/85\}$ means pivot on line 2.

max Z

$$\frac{16}{3}a - \frac{23}{15}s_c - Z = -736$$

$$\frac{1}{3}a + b + \frac{1}{15}s_c = 32$$

$$\frac{8}{3}a - \frac{4}{15}s_c + s_h = 32$$

$$\frac{85}{3}a - \frac{4}{3}s_c + s_m = 550$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{b, s_h, s_m\}$

$a = s_c = 0$

$Z = 736$

$b = 32$

$s_h = 32$

$s_m = 550$

Choose variable a to bring into basis.

Computing $\min\{3 \cdot 32, 3 \cdot 32/8, 3 \cdot 550/85\}$ means pivot on line 2.

Substitute $a = \frac{3}{8}(32 + \frac{4}{15}s_c - s_h)$.

max Z

$$\frac{16}{3}a - \frac{23}{15}s_c - Z = -736$$

$$\frac{1}{3}a + b + \frac{1}{15}s_c = 32$$

$$\frac{8}{3}a - \frac{4}{15}s_c + s_h = 32$$

$$\frac{85}{3}a - \frac{4}{3}s_c + s_m = 550$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{b, s_h, s_m\}$

$$a = s_c = 0$$

$$Z = 736$$

$$b = 32$$

$$s_h = 32$$

$$s_m = 550$$

Choose variable a to bring into basis.

Computing $\min\{3 \cdot 32, 3 \cdot 32/8, 3 \cdot 550/85\}$ means pivot on line 2.

Substitute $a = \frac{3}{8}(32 + \frac{4}{15}s_c - s_h)$.

max Z

$$-s_c - 2s_h - Z = -800$$

$$b + \frac{1}{10}s_c - \frac{1}{8}s_h = 28$$

$$a - \frac{1}{10}s_c + \frac{3}{8}s_h = 12$$

$$\frac{3}{2}s_c - \frac{85}{8}s_h + s_m = 210$$

$$a, b, s_c, s_h, s_m \geq 0$$

basis = $\{a, b, s_m\}$

$$s_c = s_h = 0$$

$$Z = 800$$

$$b = 28$$

$$a = 12$$

$$s_m = 210$$

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- If any feasible solution satisfies all constraints in the problem, then any feasible solution satisfies all constraints in the dual problem.
- If any feasible solution satisfies all constraints in the dual problem, then any feasible solution satisfies all constraints in the primal problem.
- If a primal solution value $\leq$ at most dual solution value $\leq$ at most dual solution value.
- If a dual solution value $\geq$ at least primal solution value $\geq$ at least primal solution value.

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- ▶ any feasible solution satisfies all equations in the tableaux
- ▶ in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- ▶ hence optimum solution value is at most 800
- ▶ the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- ▶ any feasible solution satisfies all equations in the tableaux
- ▶ in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
 - ▶ hence optimum solution value is at most 800
 - ▶ the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- ▶ any feasible solution satisfies all equations in the tableaux
- ▶ in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- ▶ hence optimum solution value is at most 800
- ▶ the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- ▶ any feasible solution satisfies all equations in the tableaux
- ▶ in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- ▶ hence optimum solution value is at most 800
- ▶ the current solution has value 800

Matrix View

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Matrix View

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Matrix View

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Matrix View

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

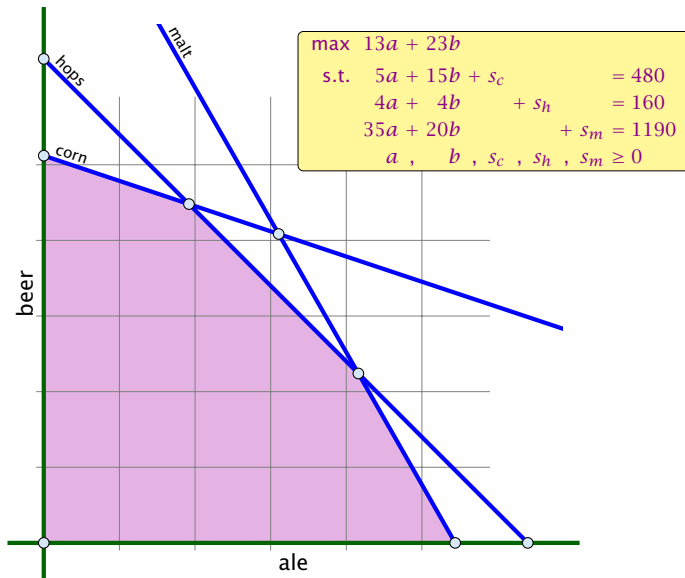
The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

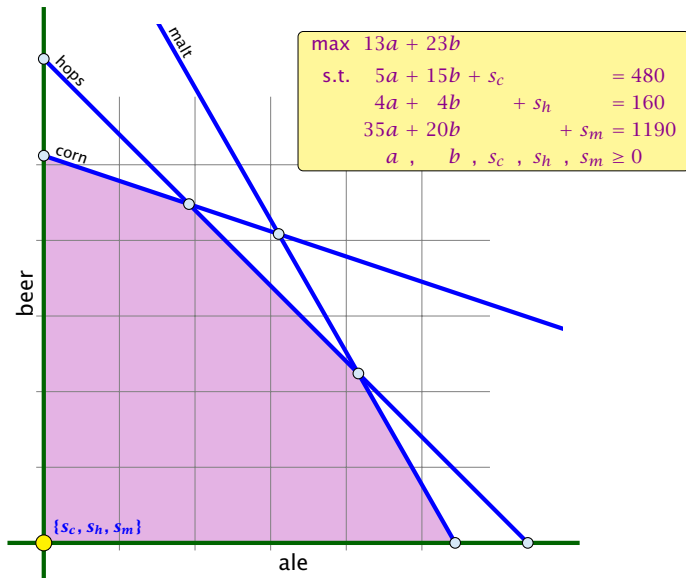
The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

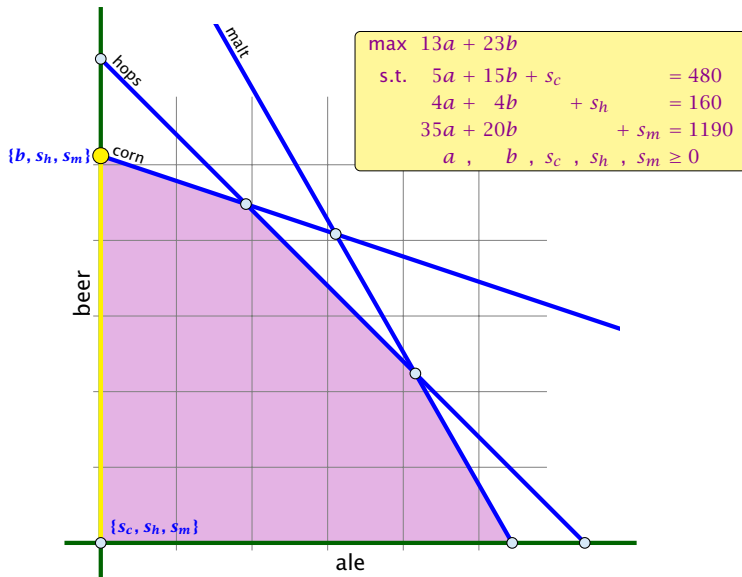
Geometric View of Pivoting



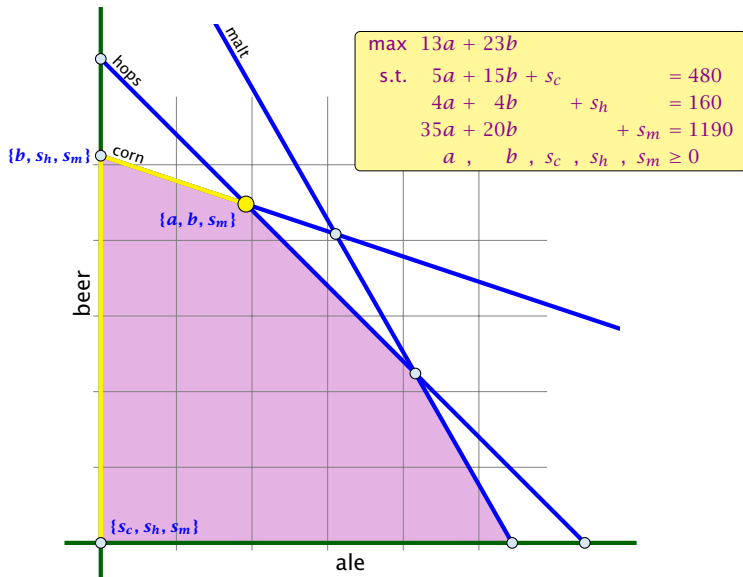
Geometric View of Pivoting



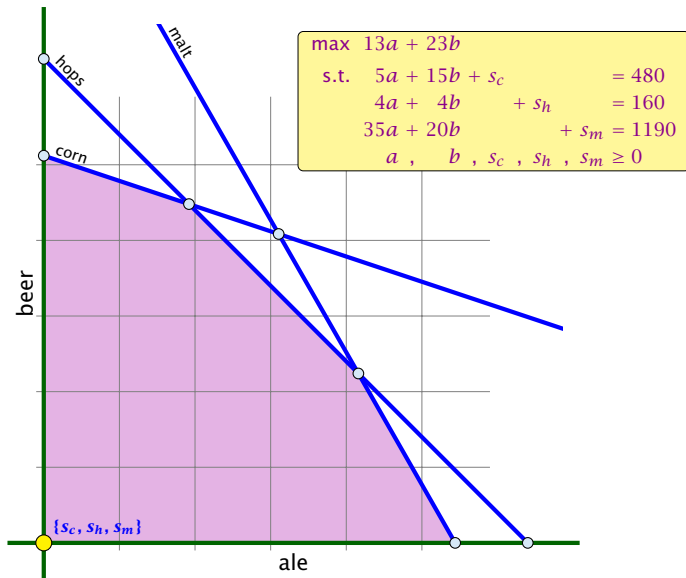
Geometric View of Pivoting



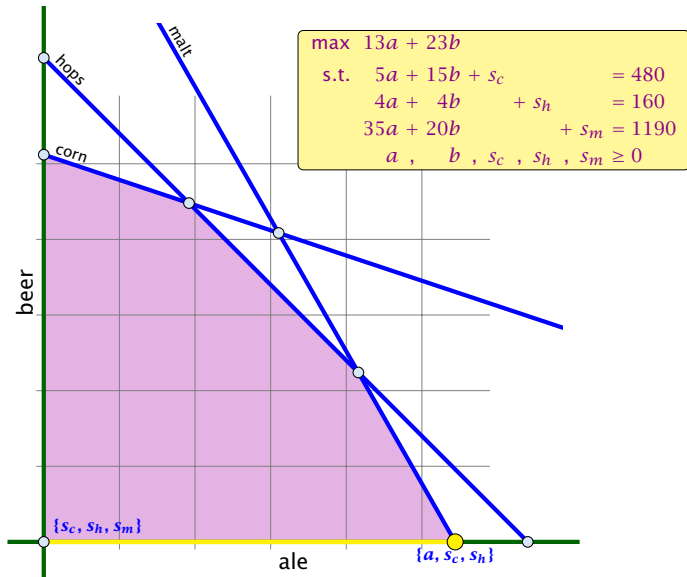
Geometric View of Pivoting



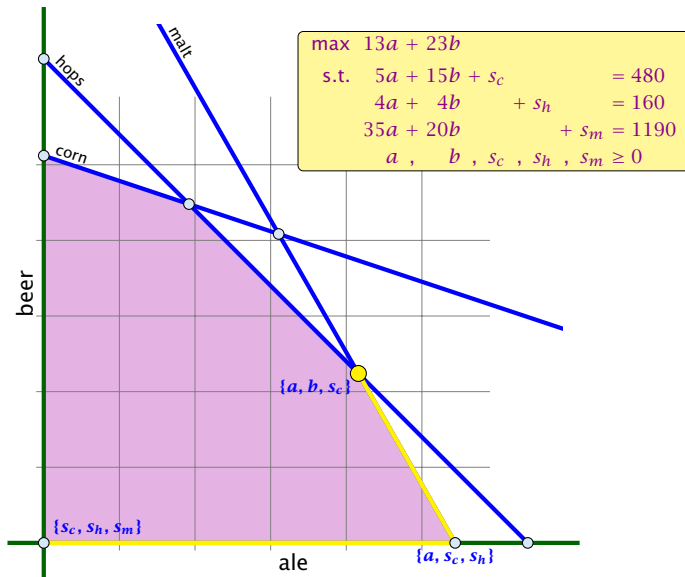
Geometric View of Pivoting



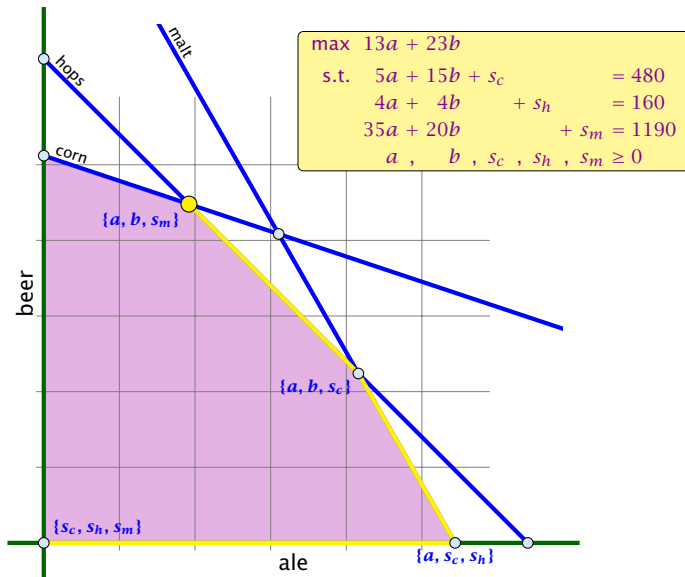
Geometric View of Pivoting



Geometric View of Pivoting



Geometric View of Pivoting



Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - Other non-basis variables should stay at 0.
 - Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

1. $d_j = 1$ (normalization)

2. $d_B = -A_B^{-1} A_j$ (feasibility, d is a direction)

3. d must hold, hence $d_B = -A_B^{-1} A_j$ must hold

4. d is a direction, $d \geq 0$, which gives

$d_B = -A_B^{-1} A_j \geq 0$

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

- ▶ $d_j = 1$ (normalization)
- ▶ $d_i = 0$ for all $i \notin B$
- ▶ $d_i \leq 0$ for all $i \in B$ (feasibility must hold, hence $\theta \cdot d_i \geq 0$)
- ▶ $d_i = 0$ for all $i \in B$ (primal feasibility must hold, hence $\theta \cdot d_i = 0$)

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

- ▶ $d_j = 1$ (normalization)
- ▶ $d_i = 0$ for all $i \notin B$
- ▶ $d_i \leq 0$ for all $i \in B$ (feasibility must hold, hence $\theta \geq 0$)
- ▶ $d_i = 0$ for all $i \in B$ (if not, the Simplex Algorithm would not terminate)

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

1. $d_j = 1$ (normalization)

2. $d_i = 0$ for all $i \in B$

3. d is a feasible direction, i.e. $d \geq 0$

4. d is the direction of the edge e_j starting at x^*

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

• $d_j = 1$ (normalization)

• $d_i = 0$ for all $i \notin B$

• $d_i = -a_{ij}$ for all $i \in B$ (maintain feasibility)

• $d_i = 0$ for all $i \in B$ with $a_{ij} = 0$ (avoid division by 0)

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

- ▶ $d_j = 1$ (normalization)
- ▶ $d_\ell = 0, \ell \notin B, \ell \neq j$
- ▶ $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- ▶ Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

- ▶ $d_j = 1$ (normalization)
- ▶ $d_\ell = 0, \ell \notin B, \ell \neq j$
- ▶ $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- ▶ Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

- ▶ $d_j = 1$ (normalization)
- ▶ $d_\ell = 0, \ell \notin B, \ell \neq j$
- ▶ $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- ▶ Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives
 $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- ▶ Given basis B with BFS x^* .
- ▶ Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
 - ▶ Other non-basis variables should stay at 0.
 - ▶ Basis variables change to maintain feasibility.
- ▶ Go from x^* to $x^* + \theta \cdot d$.

Requirements for d :

- ▶ $d_j = 1$ (normalization)
- ▶ $d_\ell = 0, \ell \notin B, \ell \neq j$
- ▶ $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- ▶ Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

Definition 26 (j -th basis direction)

Let B be a basis, and let $j \notin B$. The vector d with $d_j = 1$ and $d_\ell = 0, \ell \notin B, \ell \neq j$ and $d_B = -A_B^{-1}A_{*j}$ is called the j -th basis direction for B .

Going from x^* to $x^* + \theta \cdot d$ the objective function changes by

$$\theta \cdot c^T d = \theta(c_j - c_B^T A_B^{-1} A_{*j})$$

Algebraic Definition of Pivoting

Definition 26 (j -th basis direction)

Let B be a basis, and let $j \notin B$. The vector d with $d_j = 1$ and $d_\ell = 0, \ell \notin B, \ell \neq j$ and $d_B = -A_B^{-1}A_{*j}$ is called the j -th basis direction for B .

Going from x^* to $x^* + \theta \cdot d$ the objective function changes by

$$\theta \cdot c^T d = \theta(c_j - c_B^T A_B^{-1} A_{*j})$$

Algebraic Definition of Pivoting

Definition 27 (Reduced Cost)

For a basis B the value

$$\tilde{c}_j = c_j - c_B^T A_B^{-1} A_{*j}$$

is called the **reduced cost** for variable x_j .

Note that this is defined for every j . If $j \in B$ then the above term is 0.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

4 Simplex Algorithm

Questions:

What happens if the min ratio test fails to give us a value by which we can safely increase the entering variable?

How do we find the initial basic feasible solution?

Is there always a basis? Why/How?

What happens if the min ratio test fails to give us a value by which we can safely increase the entering variable?

Then we can terminate, because we know that the solution is unbounded. But what if the min ratio test fails to give us a value by which we can safely increase the entering variable?

Answer: If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- ▶ What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- ▶ How do we find the initial basic feasible solution?
- ▶ Is there always a basis B such that

$$(c_N^T - c_B^T A_B^{-1} A_N) \leq 0 \quad ?$$

Then we can terminate because we know that the solution is optimal.

- ▶ If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- ▶ What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- ▶ How do we find the initial basic feasible solution?
- ▶ Is there always a basis B such that

$$(c_N^T - c_B^T A_B^{-1} A_N) \leq 0 \quad ?$$

Then we can terminate because we know that the solution is optimal.

- ▶ If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- ▶ What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- ▶ How do we find the initial basic feasible solution?
- ▶ Is there always a basis B such that

$$(c_N^T - c_B^T A_B^{-1} A_N) \leq 0 \quad ?$$

Then we can terminate because we know that the solution is optimal.

- ▶ If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- ▶ What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- ▶ How do we find the initial basic feasible solution?
- ▶ Is there always a basis B such that

$$(c_N^T - c_B^T A_B^{-1} A_N) \leq 0 \quad ?$$

Then we can terminate because we know that the solution is optimal.

- ▶ If yes how do we make sure that we reach such a basis?

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this, one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b . Hence, there is no danger of this basic variable becoming negative

What happens if all b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this, one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b . Hence, there is no danger of this basic variable becoming negative

What happens if all b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this, one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b . Hence, there is no danger of this basic variable becoming negative

What happens if all b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this, one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b . Hence, there is no danger of this basic variable becoming negative

What happens if all b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this, one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b . Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this, one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b . Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. **Then the LP is unbounded!**

Termination

The objective function does not decrease during one iteration of the simplex-algorithm.

Does it always increase?

Termination

The objective function does not decrease during one iteration of the simplex-algorithm.

Does it always increase?

Termination

The objective function does not decrease during one iteration of the simplex-algorithm.

Does it always increase?

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

The set of inequalities is **degenerate** (also the basis is degenerate).

Definition 28 (Degeneracy)

A BFS x^* is called **degenerate** if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm **cycles**, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

The set of inequalities is **degenerate** (also the basis is degenerate).

Definition 28 (Degeneracy)

A BFS x^* is called **degenerate** if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm **cycles**, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

The set of inequalities is **degenerate** (also the basis is degenerate).

Definition 28 (Degeneracy)

A BFS x^* is called **degenerate** if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm **cycles**, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

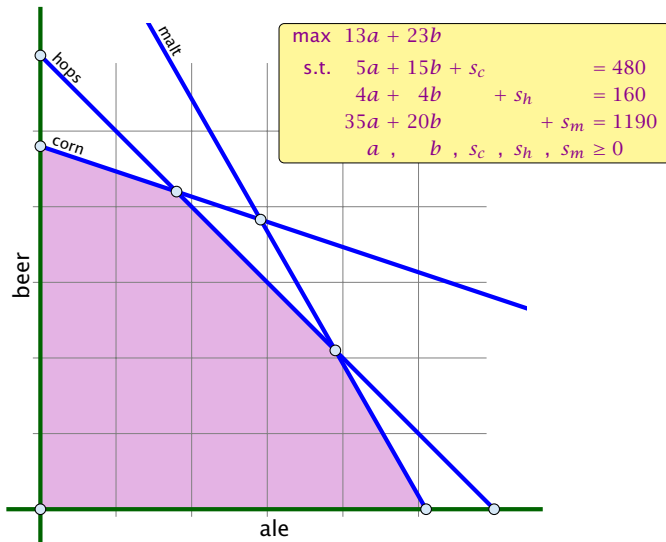
The set of inequalities is **degenerate** (also the basis is degenerate).

Definition 28 (Degeneracy)

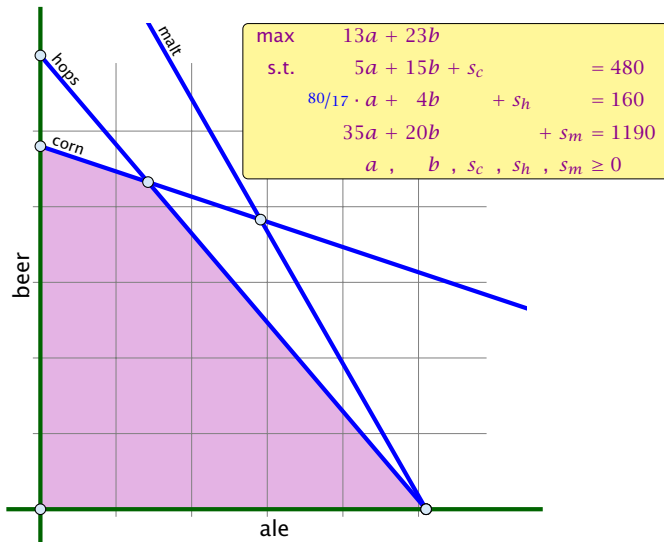
A BFS x^* is called **degenerate** if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm **cycles**, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

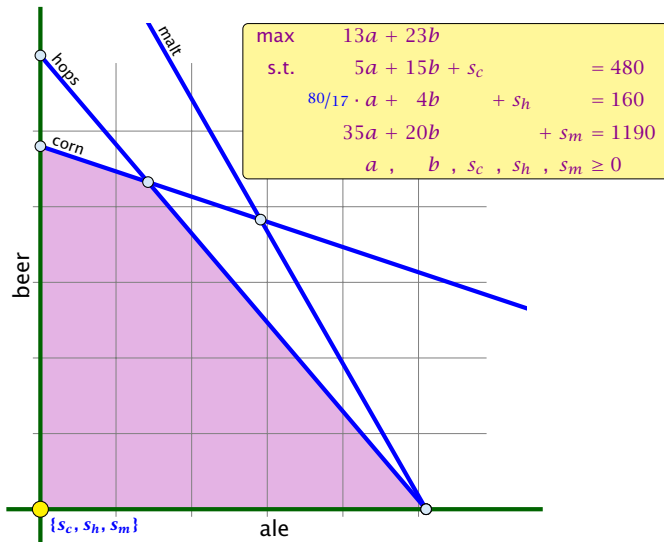
Non Degenerate Example



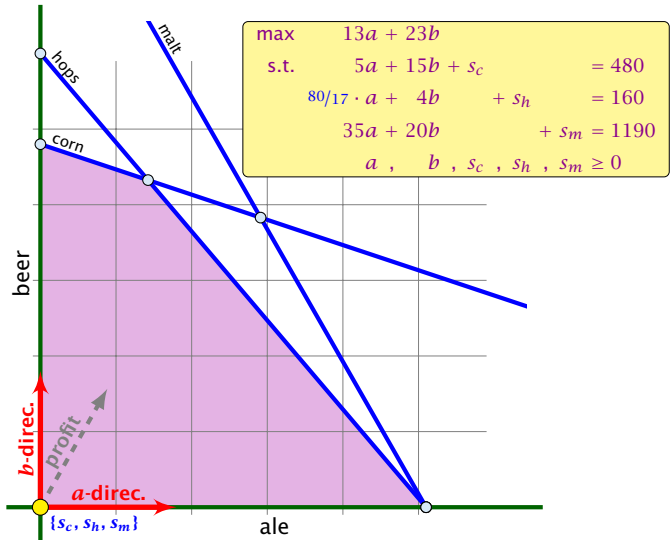
Degenerate Example



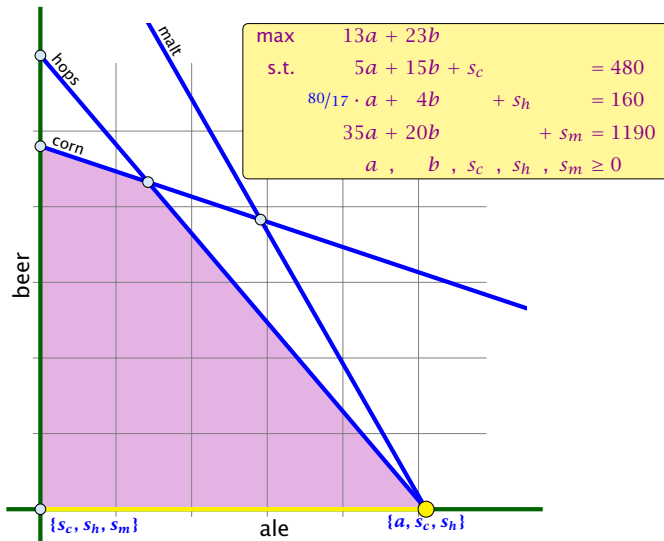
Degenerate Example



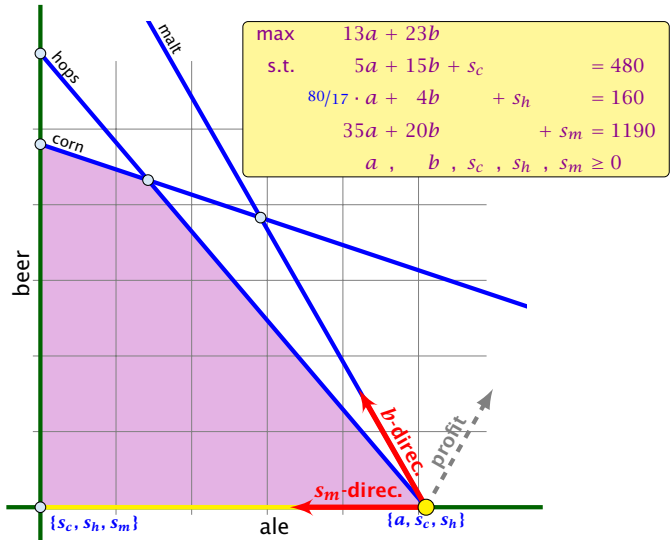
Degenerate Example



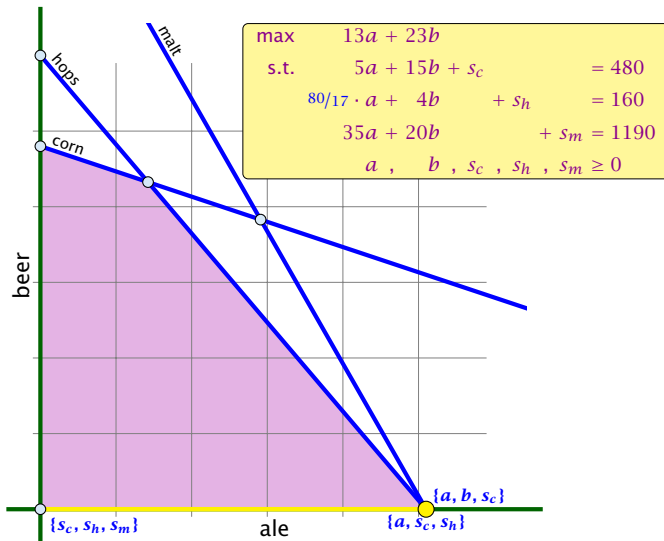
Degenerate Example



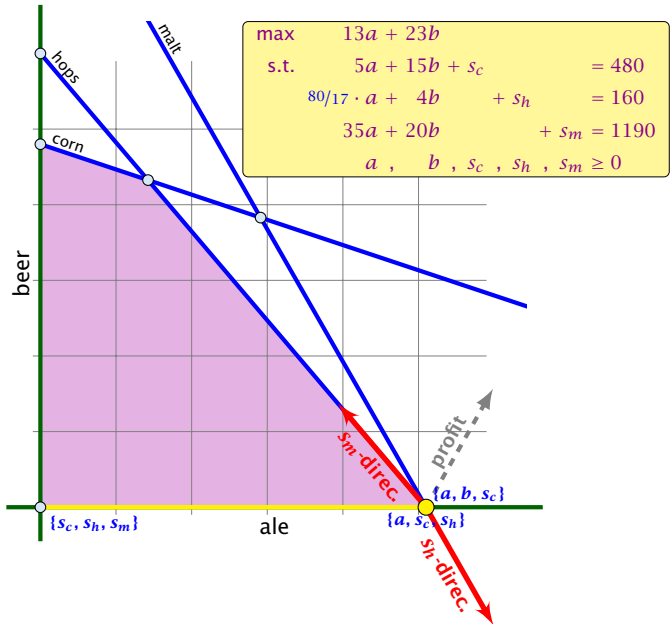
Degenerate Example



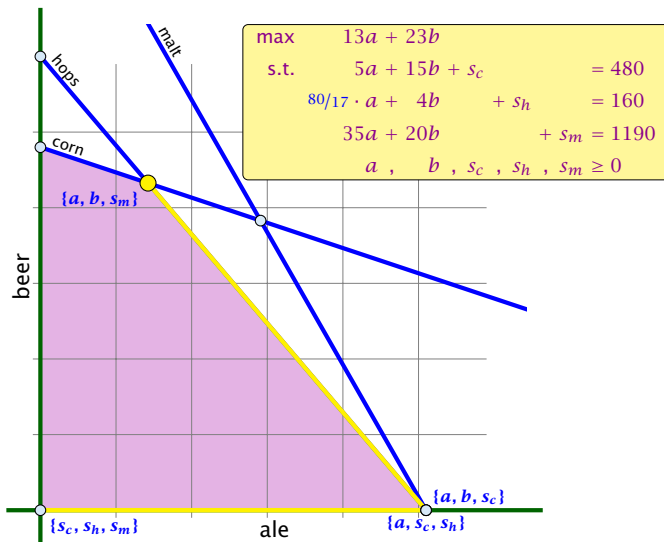
Degenerate Example



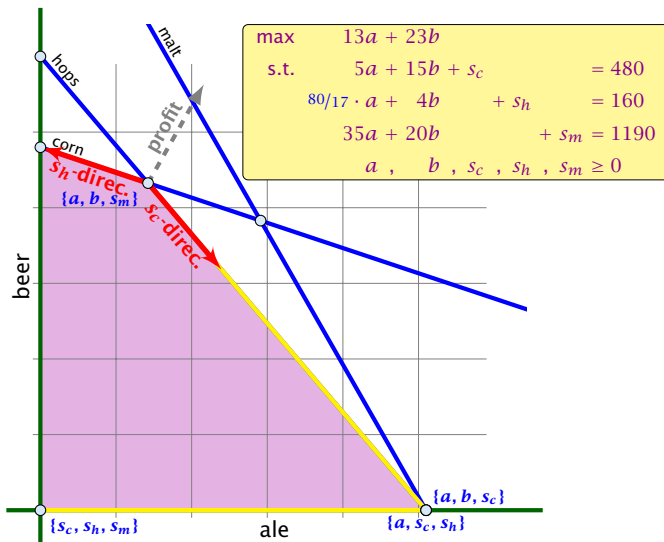
Degenerate Example



Degenerate Example



Degenerate Example



Summary: How to choose pivot-elements

- ▶ We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- ▶ The standard choice is the column that maximizes $\tilde{c}_e$.
- ▶ If $A_{ie} \leq 0$ for all $i \in \{1, \dots, m\}$ then the maximum is not bounded.
- ▶ Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- ▶ If several variables have minimum $b_\ell / A_{\ell e}$ you reach a **degenerate** basis.
- ▶ Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- ▶ We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- ▶ The standard choice is the column that maximizes $\tilde{c}_e$.
- ▶ If $A_{ie} \leq 0$ for all $i \in \{1, \dots, m\}$ then the maximum is not bounded.
- ▶ Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- ▶ If several variables have minimum $b_\ell / A_{\ell e}$ you reach a **degenerate** basis.
- ▶ Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- ▶ We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- ▶ The standard choice is the column that maximizes $\tilde{c}_e$.
- ▶ If $A_{ie} \leq 0$ for all $i \in \{1, \dots, m\}$ then the maximum is not bounded.
- ▶ Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- ▶ If several variables have minimum $b_\ell / A_{\ell e}$ you reach a **degenerate** basis.
- ▶ Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- ▶ We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- ▶ The standard choice is the column that maximizes $\tilde{c}_e$.
- ▶ If $A_{ie} \leq 0$ for all $i \in \{1, \dots, m\}$ then the maximum is not bounded.
- ▶ Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- ▶ If several variables have minimum $b_\ell / A_{\ell e}$ you reach a **degenerate** basis.
- ▶ Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- ▶ We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- ▶ The standard choice is the column that maximizes $\tilde{c}_e$.
- ▶ If $A_{ie} \leq 0$ for all $i \in \{1, \dots, m\}$ then the maximum is not bounded.
- ▶ Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- ▶ If several variables have minimum $b_\ell / A_{\ell e}$ you reach a **degenerate** basis.
- ▶ Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- ▶ We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- ▶ The standard choice is the column that maximizes $\tilde{c}_e$.
- ▶ If $A_{ie} \leq 0$ for all $i \in \{1, \dots, m\}$ then the maximum is not bounded.
- ▶ Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- ▶ If several variables have minimum $b_\ell / A_{\ell e}$ you reach a **degenerate** basis.
- ▶ Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

What do we have so far?

Suppose we are given an initial feasible solution to an LP. If the LP is non-degenerate then Simplex will terminate.

Note that we either terminate because the min-ratio test fails and we can conclude that the LP is **unbounded**, or we terminate because the vector of reduced cost is non-positive. In the latter case we have an **optimum solution**.

How do we come up with an initial solution?

- ▶ $Ax \leq b, x \geq 0$, and $b \geq 0$.
- ▶ The standard slack form for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- ▶ Then $s = b, x = 0$ is a basic feasible solution (how?).
- ▶ We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- ▶ $Ax \leq b, x \geq 0$, and $b \geq 0$.
- ▶ The standard slack form for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- ▶ Then $s = b, x = 0$ is a basic feasible solution (how?).
- ▶ We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- ▶ $Ax \leq b, x \geq 0$, and $b \geq 0$.
- ▶ The standard slack form for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- ▶ Then $s = b, x = 0$ is a basic feasible solution (how?).
- ▶ We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- ▶ $Ax \leq b, x \geq 0$, and $b \geq 0$.
- ▶ The standard slack form for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- ▶ Then $s = b, x = 0$ is a basic feasible solution (how?).
- ▶ We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- ▶ $Ax \leq b, x \geq 0$, and $b \geq 0$.
- ▶ The standard slack form for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- ▶ Then $s = b, x = 0$ is a basic feasible solution (how?).
- ▶ We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

Multiply all rows with $b_i < 0$ by -1 .

maximize $-c^T x$ s.t. $Ax = b, x \geq 0$, $b \geq 0$ using Simplex. If $b_i = 0$, $x_i = 0$ is initial feasible.

If $b_i > 0$ then the original problem is infeasible.

Otherwise you have $x = 0$ with $Ax = b$.

From this you can get basic feasible solution.

Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1 .
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is **infeasible**.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1 .
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1 .
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is **infeasible**.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1 .
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is **infeasible**.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1 .
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is **infeasible**.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^T x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1 .
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is **infeasible**.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Lemma 29

Let B be a basis and x^ a BFS corresponding to basis B . $\tilde{c} \leq 0$ implies that x^* is an optimum solution to the LP.*

Duality

How do we get an upper bound to a maximization LP?

$$\begin{array}{llll} \max & 13a & + & 23b \\ \text{s.t.} & 5a & + & 15b \leq 480 \\ & 4a & + & 4b \leq 160 \\ & 35a & + & 20b \leq 1190 \\ & & & a, b \geq 0 \end{array}$$

Note that a lower bound is easy to derive. Every choice of $a, b \geq 0$ gives us a lower bound (e.g. $a = 12, b = 28$ gives us a lower bound of 800).

If you take a conic combination of the rows (multiply the i -th row with $y_i \geq 0$) such that $\sum_i y_i a_{ij} \geq c_j$ then $\sum_i y_i b_i$ will be an upper bound.

Duality

How do we get an upper bound to a maximization LP?

$$\begin{array}{ll} \max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0 \end{array}$$

Note that a lower bound is easy to derive. Every choice of $a, b \geq 0$ gives us a lower bound (e.g. $a = 12, b = 28$ gives us a lower bound of 800).

If you take a conic combination of the rows (multiply the i -th row with $y_i \geq 0$) such that $\sum_i y_i a_{ij} \geq c_j$ then $\sum_i y_i b_i$ will be an upper bound.

Duality

How do we get an upper bound to a maximization LP?

$$\begin{array}{ll} \max & 13a + 23b \\ \text{s.t.} & 5a + 15b \leq 480 \\ & 4a + 4b \leq 160 \\ & 35a + 20b \leq 1190 \\ & a, b \geq 0 \end{array}$$

Note that a lower bound is easy to derive. Every choice of $a, b \geq 0$ gives us a lower bound (e.g. $a = 12, b = 28$ gives us a lower bound of 800).

If you take a conic combination of the rows (multiply the i -th row with $y_i \geq 0$) such that $\sum_i y_i a_{ij} \geq c_j$ then $\sum_i y_i b_i$ will be an upper bound.

Definition 30

Let $z = \max\{c^T x \mid Ax \leq b, x \geq 0\}$ be a linear program P (called the primal linear program).

The linear program D defined by

$$w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$$

is called the **dual problem**.

Lemma 31

The dual of the dual problem is the primal problem.

Proof:

The dual problem is

Lemma 31

The dual of the dual problem is the primal problem.

Proof:

$$\blacktriangleright w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$$

$$\blacktriangleright w = -\max\{-b^T y \mid -A^T y \leq -c, y \geq 0\}$$

The dual problem is

Lemma 31

The dual of the dual problem is the primal problem.

Proof:

- ▶ $w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$
- ▶ $w = -\max\{-b^T y \mid -A^T y \leq -c, y \geq 0\}$

The dual problem is

Lemma 31

The dual of the dual problem is the primal problem.

Proof:

- ▶ $w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$
- ▶ $w = -\max\{-b^T y \mid -A^T y \leq -c, y \geq 0\}$

The dual problem is

- ▶ $z = -\min\{-c^T x \mid -Ax \geq -b, x \geq 0\}$
- ▶ $z = \max\{c^T x \mid Ax \leq b, x \geq 0\}$

Lemma 31

The dual of the dual problem is the primal problem.

Proof:

- ▶ $w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$
- ▶ $w = -\max\{-b^T y \mid -A^T y \leq -c, y \geq 0\}$

The dual problem is

- ▶ $z = -\min\{-c^T x \mid -Ax \geq -b, x \geq 0\}$
- ▶ $z = \max\{c^T x \mid Ax \leq b, x \geq 0\}$

Weak Duality

Let $z = \max\{c^T x \mid Ax \leq b, x \geq 0\}$ and
 $w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$ be a primal dual pair.

x is primal feasible iff $x \in \{x \mid Ax \leq b, x \geq 0\}$

y is dual feasible, iff $y \in \{y \mid A^T y \geq c, y \geq 0\}$.

Theorem 32 (Weak Duality)

Let $\hat{x}$ be primal feasible and let $\hat{y}$ be dual feasible. Then

$$c^T \hat{x} \leq z \leq w \leq b^T \hat{y} .$$

Weak Duality

Let $z = \max\{c^T x \mid Ax \leq b, x \geq 0\}$ and
 $w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$ be a primal dual pair.

x is primal feasible iff $x \in \{x \mid Ax \leq b, x \geq 0\}$

y is dual feasible, iff $y \in \{y \mid A^T y \geq c, y \geq 0\}$.

Theorem 32 (Weak Duality)

Let $\hat{x}$ be primal feasible and let $\hat{y}$ be dual feasible. Then

$$c^T \hat{x} \leq z \leq w \leq b^T \hat{y} .$$

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq y^T b \quad (y \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A\hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq y^T b \quad (y \geq 0)$$

This gives

$$c^T \hat{x} \leq y^T A\hat{x} \leq b^T y .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq \hat{y}^T b \quad (\hat{y} \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A\hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq y^T b \quad (y \geq 0)$$

This gives

$$c^T \hat{x} \leq y^T A\hat{x} \leq b^T y .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq y^T b \quad (y \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A\hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A \hat{x} \leq b \Rightarrow y^T A \hat{x} \leq y^T b \quad (y \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A \hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq \hat{y}^T b \quad (\hat{y} \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A\hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq \hat{y}^T b \quad (\hat{y} \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A\hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^T \hat{y} \geq c \Rightarrow \hat{x}^T A^T \hat{y} \geq \hat{x}^T c \quad (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^T A\hat{x} \leq \hat{y}^T b \quad (\hat{y} \geq 0)$$

This gives

$$c^T \hat{x} \leq \hat{y}^T A\hat{x} \leq b^T \hat{y} .$$

Since, there exists primal feasible $\hat{x}$ with $c^T \hat{x} = z$, and dual feasible $\hat{y}$ with $b^T \hat{y} = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

5.2 Simplex and Duality

The following linear programs form a primal dual pair:

$$z = \max\{c^T x \mid Ax = b, x \geq 0\}$$
$$w = \min\{b^T y \mid A^T y \geq c\}$$

This means for computing the dual of a standard form LP, we do not have non-negativity constraints for the dual variables.

Proof

Primal:

$$\max\{c^T x \mid Ax = b, x \geq 0\}$$

Proof

Primal:

$$\begin{aligned}\max\{c^T x \mid Ax = b, x \geq 0\} \\ = \max\{c^T x \mid Ax \leq b, -Ax \leq -b, x \geq 0\}\end{aligned}$$

Proof

Primal:

$$\begin{aligned}\max\{c^T x \mid Ax = b, x \geq 0\} \\&= \max\{c^T x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\&= \max\{c^T x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\}\end{aligned}$$

Proof

Primal:

$$\begin{aligned} \max\{c^T x \mid Ax = b, x \geq 0\} \\ = \max\{c^T x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\ = \max\{c^T x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\} \end{aligned}$$

Dual:

$$\min\{[b^T \ -b^T]y \mid [A^T \ -A^T]y \geq c, y \geq 0\}$$

Proof

Primal:

$$\begin{aligned} & \max\{c^T x \mid Ax = b, x \geq 0\} \\ &= \max\{c^T x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\ &= \max\{c^T x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\} \end{aligned}$$

Dual:

$$\begin{aligned} & \min\{[b^T \ -b^T]y \mid [A^T \ -A^T]y \geq c, y \geq 0\} \\ &= \min\left\{[b^T \ -b^T] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \mid [A^T \ -A^T] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \geq c, y^- \geq 0, y^+ \geq 0\right\} \end{aligned}$$

Proof

Primal:

$$\begin{aligned} \max\{c^T x \mid Ax = b, x \geq 0\} \\ = \max\{c^T x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\ = \max\{c^T x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\} \end{aligned}$$

Dual:

$$\begin{aligned} \min\{[b^T \ -b^T]y \mid [A^T \ -A^T]y \geq c, y \geq 0\} \\ = \min\left\{[b^T \ -b^T] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \mid [A^T \ -A^T] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \geq c, y^- \geq 0, y^+ \geq 0\right\} \\ = \min\{b^T \cdot (y^+ - y^-) \mid A^T \cdot (y^+ - y^-) \geq c, y^- \geq 0, y^+ \geq 0\} \end{aligned}$$

Proof

Primal:

$$\begin{aligned} & \max\{c^T x \mid Ax = b, x \geq 0\} \\ &= \max\{c^T x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\ &= \max\{c^T x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\} \end{aligned}$$

Dual:

$$\begin{aligned} & \min\{[b^T \ -b^T]y \mid [A^T \ -A^T]y \geq c, y \geq 0\} \\ &= \min\left\{[b^T \ -b^T] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \mid [A^T \ -A^T] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \geq c, y^- \geq 0, y^+ \geq 0\right\} \\ &= \min\{b^T \cdot (y^+ - y^-) \mid A^T \cdot (y^+ - y^-) \geq c, y^- \geq 0, y^+ \geq 0\} \\ &= \min\{b^T y' \mid A^T y' \geq c\} \end{aligned}$$

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

Since y^* is feasible and $y^* b = c^T x^*$, the solution is optimal.

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with **reduced cost**

$$\tilde{c} = c^T - c_B^T A_B^{-1} A \leq 0$$

This is equivalent to $A^T (A_B^{-1})^T c_B \geq c$

$y^* = (A_B^{-1})^T c_B$ is solution to the **dual** $\min\{b^T y \mid A^T y \geq c\}$.

$$\begin{aligned} b^T y^* &= (Ax^*)^T y^* = (A_B x_B^*)^T y^* \\ &= (A_B x_B^*)^T (A_B^{-1})^T c_B = (x_B^*)^T A_B^T (A_B^{-1})^T c_B \\ &= c^T x^* \end{aligned}$$

Hence, the solution is optimal.

5.3 Strong Duality

$$P = \max\{c^T x \mid Ax \leq b, x \geq 0\}$$

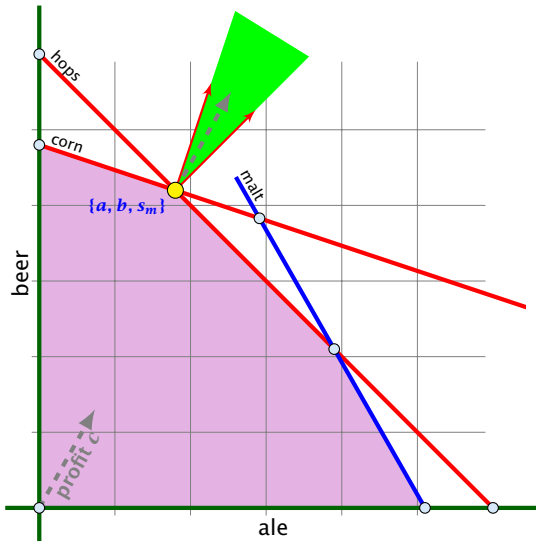
n_A : number of variables, m_A : number of constraints

We can put the non-negativity constraints into A (which gives us unrestricted variables): $\bar{P} = \max\{c^T x \mid \bar{A}x \leq \bar{b}\}$

$$n_{\bar{A}} = n_A, m_{\bar{A}} = m_A + n_A$$

$$\text{Dual } D = \min\{\bar{b}^T y \mid \bar{A}^T y = c, y \geq 0\}.$$

5.3 Strong Duality



The profit vector c lies in the cone generated by the normals for the hops and the corn constraint (the tight constraints).

Strong Duality

Theorem 33 (Strong Duality)

Let P and D be a primal dual pair of linear programs, and let z^* and w^* denote the optimal solution to P and D , respectively. Then

$$z^* = w^*$$

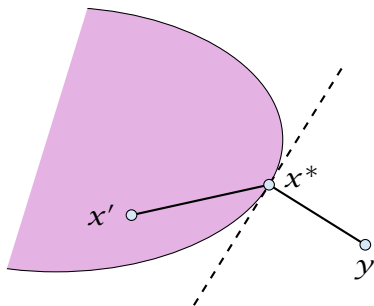
Lemma 34 (Weierstrass)

Let X be a compact set and let $f(x)$ be a continuous function on X . Then $\min\{f(x) : x \in X\}$ exists.

(without proof)

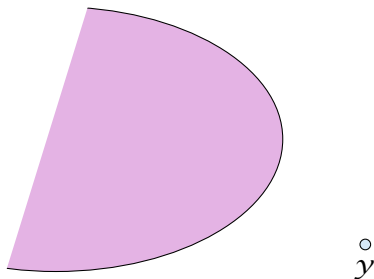
Lemma 35 (Projection Lemma)

Let $X \subseteq \mathbb{R}^m$ be a non-empty convex set, and let $y \notin X$. Then there exist $x^* \in X$ with minimum distance from y . Moreover for all $x \in X$ we have $(y - x^*)^T(x - x^*) \leq 0$.



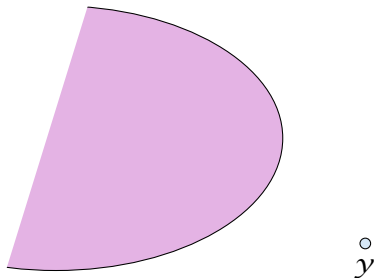
Proof of the Projection Lemma

- ▶ Define $f(x) = \|y - x\|$.
- ▶ We want to apply Weierstrass but X may not be bounded.
- ▶ $X \neq \emptyset$. Hence, there exists $x' \in X$.
- ▶ Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- ▶ Applying Weierstrass gives the existence.



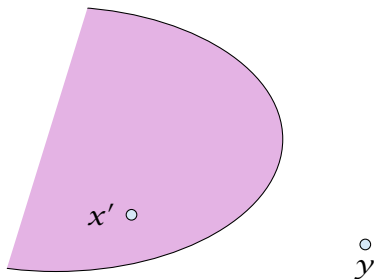
Proof of the Projection Lemma

- ▶ Define $f(x) = \|y - x\|$.
- ▶ We want to apply Weierstrass but X may not be bounded.
- ▶ $X \neq \emptyset$. Hence, there exists $x' \in X$.
- ▶ Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- ▶ Applying Weierstrass gives the existence.



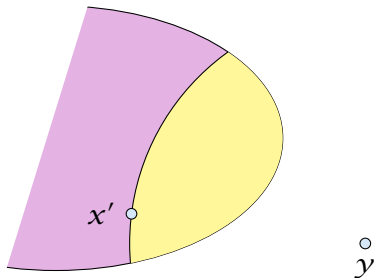
Proof of the Projection Lemma

- ▶ Define $f(x) = \|y - x\|$.
- ▶ We want to apply Weierstrass but X may not be bounded.
- ▶ $X \neq \emptyset$. Hence, there exists $x' \in X$.
- ▶ Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- ▶ Applying Weierstrass gives the existence.



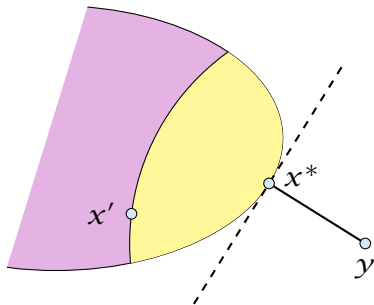
Proof of the Projection Lemma

- ▶ Define $f(x) = \|y - x\|$.
- ▶ We want to apply Weierstrass but X may not be bounded.
- ▶ $X \neq \emptyset$. Hence, there exists $x' \in X$.
- ▶ Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- ▶ Applying Weierstrass gives the existence.



Proof of the Projection Lemma

- ▶ Define $f(x) = \|y - x\|$.
- ▶ We want to apply Weierstrass but X may not be bounded.
- ▶ $X \neq \emptyset$. Hence, there exists $x' \in X$.
- ▶ Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- ▶ Applying Weierstrass gives the existence.



Proof of the Projection Lemma (continued)

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By **convexity**: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By **convexity**: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\|y - x^*\|^2$$

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By **convexity**: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\|y - x^*\|^2 \leq \|y - x^* - \epsilon(x - x^*)\|^2$$

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By **convexity**: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\begin{aligned}\|y - x^*\|^2 &\leq \|y - x^* - \epsilon(x - x^*)\|^2 \\ &= \|y - x^*\|^2 + \epsilon^2 \|x - x^*\|^2 - 2\epsilon(y - x^*)^T(x - x^*)\end{aligned}$$

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By **convexity**: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\begin{aligned}\|y - x^*\|^2 &\leq \|y - x^* - \epsilon(x - x^*)\|^2 \\ &= \|y - x^*\|^2 + \epsilon^2 \|x - x^*\|^2 - 2\epsilon(y - x^*)^T(x - x^*)\end{aligned}$$

Hence, $(y - x^*)^T(x - x^*) \leq \frac{1}{2}\epsilon\|x - x^*\|^2$.

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By **convexity**: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\begin{aligned}\|y - x^*\|^2 &\leq \|y - x^* - \epsilon(x - x^*)\|^2 \\ &= \|y - x^*\|^2 + \epsilon^2 \|x - x^*\|^2 - 2\epsilon(y - x^*)^T(x - x^*)\end{aligned}$$

Hence, $(y - x^*)^T(x - x^*) \leq \frac{1}{2}\epsilon \|x - x^*\|^2$.

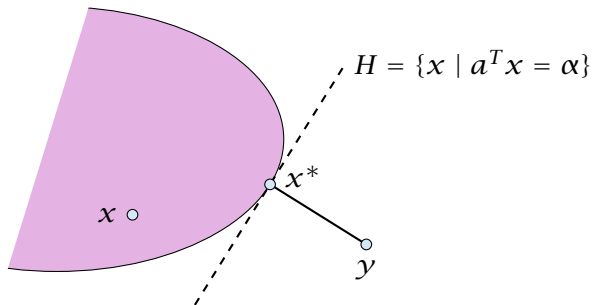
Letting $\epsilon \rightarrow 0$ gives the result.

Theorem 36 (Separating Hyperplane)

Let $X \subseteq \mathbb{R}^m$ be a non-empty closed convex set, and let $y \notin X$. Then there exists a separating hyperplane $\{x \in \mathbb{R} : a^T x = \alpha\}$ where $a \in \mathbb{R}^m$, $\alpha \in \mathbb{R}$ that separates y from X . ($a^T y < \alpha$; $a^T x \geq \alpha$ for all $x \in X$)

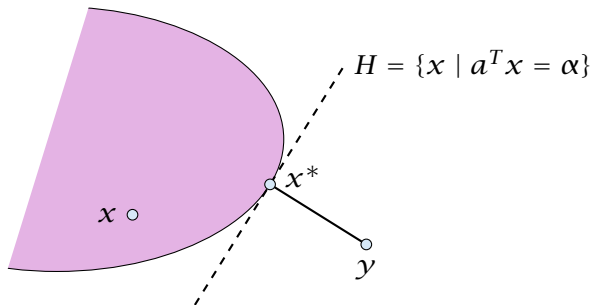
Proof of the Hyperplane Lemma

- ▶ Let $x^* \in X$ be closest point to y in X .
- ▶ By previous lemma $(y - x^*)^T(x - x^*) \leq 0$ for all $x \in X$.
- ▶ Choose $a = (x^* - y)$ and $\alpha = a^T x^*$.
- ▶ For $x \in X$: $a^T(x - x^*) \geq 0$, and, hence, $a^T x \geq \alpha$.
- ▶ Also, $a^T y = a^T(x^* - a) = \alpha - \|a\|^2 < \alpha$



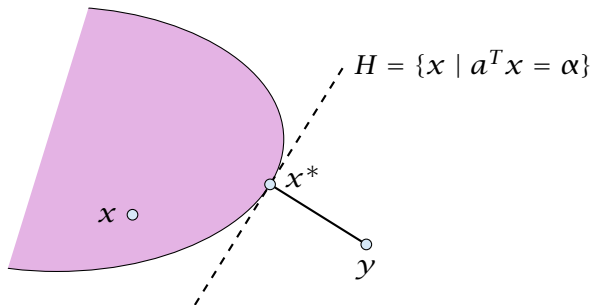
Proof of the Hyperplane Lemma

- ▶ Let $x^* \in X$ be closest point to y in X .
- ▶ By previous lemma $(y - x^*)^T(x - x^*) \leq 0$ for all $x \in X$.
- ▶ Choose $a = (x^* - y)$ and $\alpha = a^T x^*$.
- ▶ For $x \in X$: $a^T(x - x^*) \geq 0$, and, hence, $a^T x \geq \alpha$.
- ▶ Also, $a^T y = a^T(x^* - a) = \alpha - \|a\|^2 < \alpha$



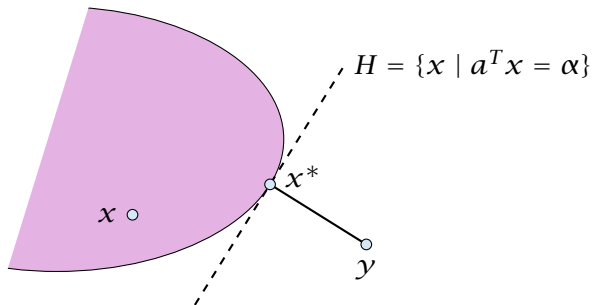
Proof of the Hyperplane Lemma

- ▶ Let $x^* \in X$ be closest point to y in X .
- ▶ By previous lemma $(y - x^*)^T(x - x^*) \leq 0$ for all $x \in X$.
- ▶ Choose $a = (x^* - y)$ and $\alpha = a^T x^*$.
- ▶ For $x \in X$: $a^T(x - x^*) \geq 0$, and, hence, $a^T x \geq \alpha$.
- ▶ Also, $a^T y = a^T(x^* - a) = \alpha - \|a\|^2 < \alpha$



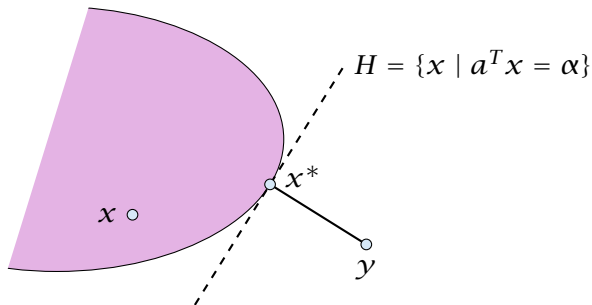
Proof of the Hyperplane Lemma

- ▶ Let $x^* \in X$ be closest point to y in X .
- ▶ By previous lemma $(y - x^*)^T(x - x^*) \leq 0$ for all $x \in X$.
- ▶ Choose $a = (x^* - y)$ and $\alpha = a^T x^*$.
- ▶ For $x \in X : a^T(x - x^*) \geq 0$, and, hence, $a^T x \geq \alpha$.
- ▶ Also, $a^T y = a^T(x^* - a) = \alpha - \|a\|^2 < \alpha$



Proof of the Hyperplane Lemma

- ▶ Let $x^* \in X$ be closest point to y in X .
- ▶ By previous lemma $(y - x^*)^T(x - x^*) \leq 0$ for all $x \in X$.
- ▶ Choose $a = (x^* - y)$ and $\alpha = a^T x^*$.
- ▶ For $x \in X : a^T(x - x^*) \geq 0$, and, hence, $a^T x \geq \alpha$.
- ▶ Also, $a^T y = a^T(x^* - a) = \alpha - \|a\|^2 < \alpha$



Lemma 37 (Farkas Lemma)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then *exactly one* of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ with $Ax = b, x \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $A^T y \geq 0, b^T y < 0$

Assume $\hat{x}$ satisfies 1. and $\hat{y}$ satisfies 2. Then

$$0 > \hat{y}^T b = \hat{y}^T A \hat{x} \geq 0$$

Hence, at most one of the statements can hold.

Lemma 37 (Farkas Lemma)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then *exactly one* of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ with $Ax = b, x \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $A^T y \geq 0, b^T y < 0$

Assume $\hat{x}$ satisfies 1. and $\hat{y}$ satisfies 2. Then

$$0 > y^T b = y^T Ax \geq 0$$

Hence, at most one of the statements can hold.

Lemma 37 (Farkas Lemma)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then *exactly one* of the following statements holds.

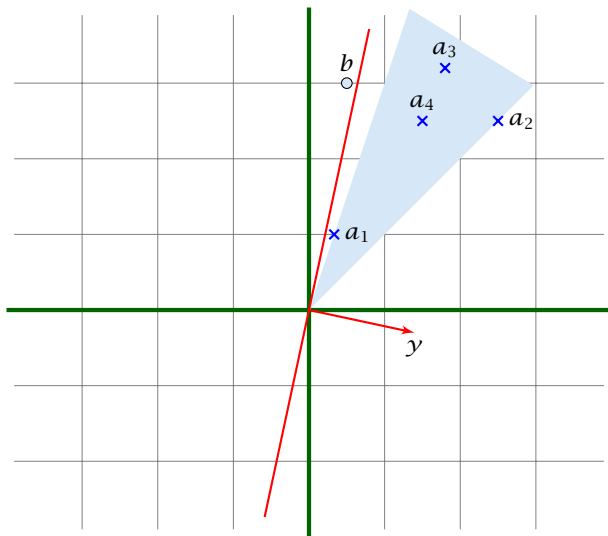
1. $\exists x \in \mathbb{R}^n$ with $Ax = b, x \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $A^T y \geq 0, b^T y < 0$

Assume $\hat{x}$ satisfies 1. and $\hat{y}$ satisfies 2. Then

$$0 > \hat{y}^T b = \hat{y}^T A \hat{x} \geq 0$$

Hence, at most one of the statements can hold.

Farkas Lemma



If b is not in the cone generated by the columns of A , there exists a hyperplane y that separates b from the cone.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^T y \geq 0$, $b^T y < 0$.

Let y be a hyperplane that separates b from S . Hence, $y^T b < \alpha$ and $y^T s \geq \alpha$ for all $s \in S$.

$$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^T b < 0$$

$y^T Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^T A \geq 0$ as we can choose x arbitrarily large.

Lemma 38 (Farkas Lemma; different version)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ with $Ax \leq b, x \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $A^T y \geq 0, b^T y < 0, y \geq 0$

Rewrite the conditions:

1. $\exists x \in \mathbb{R}^n$ with $\begin{bmatrix} A & I \end{bmatrix} \cdot \begin{bmatrix} x \\ s \end{bmatrix} = b, x \geq 0, s \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $\begin{bmatrix} A^T \\ I \end{bmatrix} y \geq 0, b^T y < 0$

Lemma 38 (Farkas Lemma; different version)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ with $Ax \leq b, x \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $A^T y \geq 0, b^T y < 0, y \geq 0$

Rewrite the conditions:

1. $\exists x \in \mathbb{R}^n$ with $\begin{bmatrix} A & I \end{bmatrix} \cdot \begin{bmatrix} x \\ s \end{bmatrix} = b, x \geq 0, s \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $\begin{bmatrix} A^T \\ I \end{bmatrix} y \geq 0, b^T y < 0$

Proof of Strong Duality

$$P: z = \max\{c^T x \mid Ax \leq b, x \geq 0\}$$

$$D: w = \min\{b^T y \mid A^T y \geq c, y \geq 0\}$$

Theorem 39 (Strong Duality)

Let P and D be a primal dual pair of linear programs, and let z and w denote the optimal solution to P and D , respectively (i.e., P and D are non-empty). Then

$$z = w \text{ .}$$

Proof of Strong Duality

Proof of Strong Duality

$z \leq w$: follows from weak duality

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:

We show $z < \alpha$ implies $w < \alpha$.

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:

We show $z < \alpha$ implies $w < \alpha$.

$$\begin{array}{lll} \exists x \in \mathbb{R}^n & & \\ \text{s.t.} & Ax & \leq b \\ & -c^T x & \leq -\alpha \\ & x & \geq 0 \end{array}$$

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:

We show $z < \alpha$ implies $w < \alpha$.

$$\exists x \in \mathbb{R}^n$$

$$\begin{array}{lll} \text{s.t.} & Ax & \leq b \\ & -c^T x & \leq -\alpha \\ & x & \geq 0 \end{array}$$

$$\exists y \in \mathbb{R}^m; v \in \mathbb{R}$$

$$\begin{array}{lll} \text{s.t.} & A^T y - cv & \geq 0 \\ & b^T y - \alpha v & < 0 \\ & y, v & \geq 0 \end{array}$$

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:

We show $z < \alpha$ implies $w < \alpha$.

$$\exists x \in \mathbb{R}^n$$

$$\begin{aligned} \text{s.t.} \quad Ax &\leq b \\ -c^T x &\leq -\alpha \\ x &\geq 0 \end{aligned}$$

$$\exists y \in \mathbb{R}^m; v \in \mathbb{R}$$

$$\begin{aligned} \text{s.t.} \quad A^T y - cv &\geq 0 \\ b^T y - \alpha v &< 0 \\ y, v &\geq 0 \end{aligned}$$

From the definition of α we know that the first system is infeasible; hence the second must be feasible.

Proof of Strong Duality

$$\begin{array}{ll}\exists y \in \mathbb{R}^m; v \in \mathbb{R} & \\ \text{s.t.} & A^T y - cv \geq 0 \\ & b^T y - \alpha v < 0 \\ & y, v \geq 0\end{array}$$

Proof of Strong Duality

$$\begin{array}{ll}\exists \mathbf{y} \in \mathbb{R}^m; v \in \mathbb{R} & \\ \text{s.t.} & A^T \mathbf{y} - c v \geq 0 \\ & b^T \mathbf{y} - \alpha v < 0 \\ & \mathbf{y}, v \geq 0\end{array}$$

If the solution $\mathbf{y}, v$ has $v = 0$ we have that

$$\begin{array}{ll}\exists \mathbf{y} \in \mathbb{R}^m & \\ \text{s.t.} & A^T \mathbf{y} \geq 0 \\ & b^T \mathbf{y} < 0 \\ & \mathbf{y} \geq 0\end{array}$$

is feasible.

Proof of Strong Duality

$$\begin{array}{ll}\exists \mathbf{y} \in \mathbb{R}^m; v \in \mathbb{R} & \\ \text{s.t.} & A^T \mathbf{y} - c v \geq 0 \\ & b^T \mathbf{y} - \alpha v < 0 \\ & \mathbf{y}, v \geq 0\end{array}$$

If the solution $\mathbf{y}, v$ has $v = 0$ we have that

$$\begin{array}{ll}\exists \mathbf{y} \in \mathbb{R}^m & \\ \text{s.t.} & A^T \mathbf{y} \geq 0 \\ & b^T \mathbf{y} < 0 \\ & \mathbf{y} \geq 0\end{array}$$

is feasible. By Farkas lemma this gives that LP P is infeasible.
Contradiction to the assumption of the lemma.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^T y < \alpha$. This means that $w < \alpha$.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^T y < \alpha$. This means that $w < \alpha$.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^T y < \alpha$. This means that $w < \alpha$.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^T y < \alpha$. This means that $w < \alpha$.

Fundamental Questions

Definition 40 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP? yes!
- ▶ Is LP in P?

Proof:

Fundamental Questions

Definition 40 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP? yes!
- ▶ Is LP in P?

Proof:

- ▶ Given a primal maximization problem P and a parameter α . Suppose that $\alpha > \text{opt}(P)$.
 - ▶ We can prove this by providing an optimal basis for the dual.
 - ▶ A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Fundamental Questions

Definition 40 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP? yes!
- ▶ Is LP in P?

Proof:

- ▶ Given a primal maximization problem P and a parameter α . Suppose that $\alpha > \text{opt}(P)$.
- ▶ We can prove this by providing an optimal basis for the dual.
- ▶ A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Fundamental Questions

Definition 40 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^T x \geq \alpha$?

Questions:

- ▶ Is LP in NP?
- ▶ Is LP in co-NP? yes!
- ▶ Is LP in P?

Proof:

- ▶ Given a primal maximization problem P and a parameter α . Suppose that $\alpha > \text{opt}(P)$.
- ▶ We can prove this by providing an optimal basis for the dual.
- ▶ A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Complementary Slackness

Lemma 41

Assume a linear program $P = \max\{c^T x \mid Ax \leq b; x \geq 0\}$ has solution x^* and its dual $D = \min\{b^T y \mid A^T y \geq c; y \geq 0\}$ has solution y^* .

1. If $x_j^* > 0$ then the j -th constraint in D is tight.
2. If the j -th constraint in D is not tight then $x_j^* = 0$.
3. If $y_i^* > 0$ then the i -th constraint in P is tight.
4. If the i -th constraint in P is not tight then $y_i^* = 0$.

Complementary Slackness

Lemma 41

Assume a linear program $P = \max\{c^T x \mid Ax \leq b; x \geq 0\}$ has solution x^* and its dual $D = \min\{b^T y \mid A^T y \geq c; y \geq 0\}$ has solution y^* .

1. If $x_j^* > 0$ then the j -th constraint in D is tight.
2. If the j -th constraint in D is not tight then $x_j^* = 0$.
3. If $y_i^* > 0$ then the i -th constraint in P is tight.
4. If the i -th constraint in P is not tight then $y_i^* = 0$.

If we say that a variable x_j^* (y_i^*) has slack if $x_j^* > 0$ ($y_i^* > 0$), (i.e., the corresponding variable restriction is not tight) and a constraint has slack if it is not tight, then the above says that for a primal-dual solution pair it is not possible that a constraint **and** its corresponding (dual) variable has slack.

Proof: Complementary Slackness

Analogous to the proof of weak duality we obtain

$$c^T x^* \leq y^{*T} A x^* \leq b^T y^*$$

Proof: Complementary Slackness

Analogous to the proof of weak duality we obtain

$$c^T x^* \leq y^{*T} A x^* \leq b^T y^*$$

Because of strong duality we then get

$$c^T x^* = y^{*T} A x^* = b^T y^*$$

This gives e.g.

$$\sum_j (y^T A - c^T)_j x_j^* = 0$$

Proof: Complementary Slackness

Analogous to the proof of weak duality we obtain

$$c^T x^* \leq y^{*T} A x^* \leq b^T y^*$$

Because of strong duality we then get

$$c^T x^* = y^{*T} A x^* = b^T y^*$$

This gives e.g.

$$\sum_j (y^T A - c^T)_j x_j^* = 0$$

From the constraint of the dual it follows that $y^T A \geq c^T$. Hence the left hand side is a sum over the product of non-negative numbers. Hence, if e.g. $(y^T A - c^T)_j > 0$ (the j -th constraint in the dual is not tight) then $x_j^* = 0$ (2.). The result for (1./3./4.) follows similarly.

Interpretation of Dual Variables

- Brewer: find mix of ale and beer that maximizes profits

$$\begin{array}{lll} \max & 13a & + \quad 23b \\ \text{s.t.} & 5a & + \quad 15b \leq 480 \\ & 4a & + \quad 4b \leq 160 \\ & 35a & + \quad 20b \leq 1190 \\ & & a, b \geq 0 \end{array}$$

- Entrepreneur: buy resources from brewer at minimum cost
 C, H, M : unit price for corn, hops and malt.

$$\begin{array}{lll} \min & 480C & + \quad 160H \quad + \quad 1190M \\ \text{s.t.} & 5C & + \quad 4H \quad + \quad 35M \geq 13 \\ & 15C & + \quad 4H \quad + \quad 20M \geq 23 \\ & & C, H, M \geq 0 \end{array}$$

Note that brewer won't sell (at least not all) if e.g.

$5C + 4H + 35M < 13$ as then brewing ale would be advantageous.

Interpretation of Dual Variables

- Brewer: find mix of ale and beer that maximizes profits

$$\begin{array}{llll} \max & 13a & + & 23b \\ \text{s.t.} & 5a & + & 15b \leq 480 \\ & 4a & + & 4b \leq 160 \\ & 35a & + & 20b \leq 1190 \\ & & & a, b \geq 0 \end{array}$$

- Entrepreneur: buy resources from brewer at minimum cost
 C, H, M : unit price for corn, hops and malt.

$$\begin{array}{llllll} \min & 480C & + & 160H & + & 1190M \\ \text{s.t.} & 5C & + & 4H & + & 35M \geq 13 \\ & 15C & + & 4H & + & 20M \geq 23 \\ & & & & & C, H, M \geq 0 \end{array}$$

Note that brewer won't sell (at least not all) if e.g.
 $5C + 4H + 35M < 13$ as then brewing ale would be advantageous.

Interpretation of Dual Variables

- Brewer: find mix of ale and beer that maximizes profits

$$\begin{array}{lll} \max & 13a & + \quad 23b \\ \text{s.t.} & 5a & + \quad 15b \leq 480 \\ & 4a & + \quad 4b \leq 160 \\ & 35a & + \quad 20b \leq 1190 \\ & & a, b \geq 0 \end{array}$$

- Entrepreneur: buy resources from brewer at minimum cost
 C, H, M : unit price for corn, hops and malt.

$$\begin{array}{llll} \min & 480C & + \quad 160H & + \quad 1190M \\ \text{s.t.} & 5C & + \quad 4H & + \quad 35M \geq 13 \\ & 15C & + \quad 4H & + \quad 20M \geq 23 \\ & & & C, H, M \geq 0 \end{array}$$

Note that brewer won't sell (at least not all) if e.g.

$5C + 4H + 35M < 13$ as then brewing ale would be advantageous.

Interpretation of Dual Variables

Marginal Price:

- ▶ How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- ▶ We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ϵ_C , ϵ_H , and ϵ_M , respectively.

The profit increases to $\max\{c^T x \mid Ax \leq b + \epsilon; x \geq 0\}$. Because of strong duality this is equal to

$$\begin{array}{ll} \min & (b^T + \epsilon^T)y \\ \text{s.t.} & A^T y \geq c \\ & y \geq 0 \end{array}$$

Interpretation of Dual Variables

Marginal Price:

- ▶ How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- ▶ We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ϵ_C , ϵ_H , and ϵ_M , respectively.

The profit increases to $\max\{c^T x \mid Ax \leq b + \epsilon; x \geq 0\}$. Because of strong duality this is equal to

$$\begin{array}{ll} \min & (b^T + \epsilon^T)y \\ \text{s.t.} & A^T y \geq c \\ & y \geq 0 \end{array}$$

Interpretation of Dual Variables

Marginal Price:

- ▶ How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- ▶ We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ϵ_C , ϵ_H , and ϵ_M , respectively.

The profit increases to $\max\{c^T x \mid Ax \leq b + \epsilon; x \geq 0\}$. Because of strong duality this is equal to

$$\begin{array}{ll} \min & (b^T + \epsilon^T)y \\ \text{s.t.} & A^T y \geq c \\ & y \geq 0 \end{array}$$

Interpretation of Dual Variables

Marginal Price:

- ▶ How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- ▶ We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ϵ_C , ϵ_H , and ϵ_M , respectively.

The profit increases to $\max\{c^T x \mid Ax \leq b + \epsilon; x \geq 0\}$. Because of strong duality this is equal to

$$\begin{array}{ll} \min & (b^T + \epsilon^T)y \\ \text{s.t.} & A^T y \geq c \\ & y \geq 0 \end{array}$$

Interpretation of Dual Variables

If ϵ is “small” enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \epsilon_i y_i^*$.

Therefore we can interpret the dual variables as **marginal prices**.

Note that with this interpretation, complementary slackness becomes obvious.

• If the buyer has slack of some resource (e.g. food) then he is not willing to pay anything for it (corresponding dual variable is zero).

• If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the buyer. Hence, it makes no sense to have leftovers of this resource. Therefore its slack must be zero.

Interpretation of Dual Variables

If ϵ is “small” enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \epsilon_i y_i^*$.

Therefore we can interpret the dual variables as **marginal prices**.

Note that with this interpretation, complementary slackness becomes obvious.

• If the buyer has slack of some resource (e.g. food) then he is not willing to pay anything for it (corresponding dual variable is zero).

• If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the buyer. Hence, it makes no sense to have an excess of this resource. Therefore slack must be zero.

Interpretation of Dual Variables

If ϵ is “small” enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \epsilon_i y_i^*$.

Therefore we can interpret the dual variables as **marginal prices**.

Note that with this interpretation, complementary slackness becomes obvious.

If the primal has slack on some resource (e.g. only 1000 lbs of flour) then the primal is not willing to pay anything for it (corresponding dual variable is zero). If the primal is tight (e.g. 1000 lbs of flour is needed) then the primal is willing to pay the corresponding dual value for it.

If we increase the right-hand side of a constraint (e.g. increase the amount of flour) then the primal is willing to pay the corresponding dual value for it.

If we decrease the right-hand side of a constraint (e.g. decrease the amount of flour) then the primal is willing to pay the corresponding dual value for it.

Interpretation of Dual Variables

If ϵ is “small” enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \epsilon_i y_i^*$.

Therefore we can interpret the dual variables as **marginal prices**.

Note that with this interpretation, complementary slackness becomes obvious.

- ▶ If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- ▶ If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

Interpretation of Dual Variables

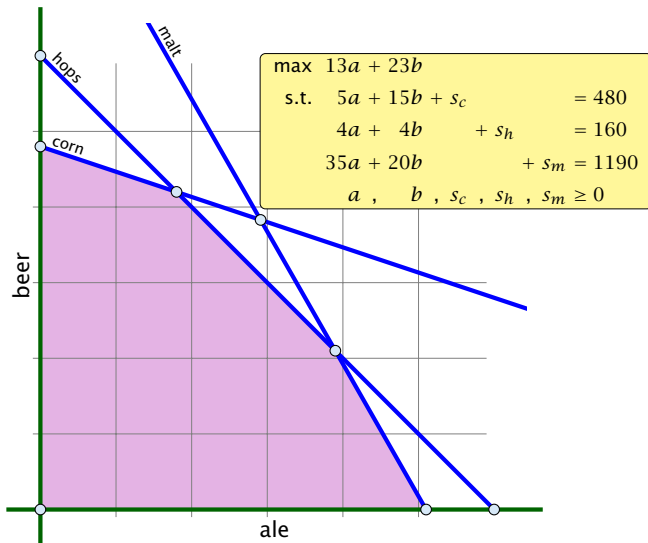
If ϵ is “small” enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \epsilon_i y_i^*$.

Therefore we can interpret the dual variables as **marginal prices**.

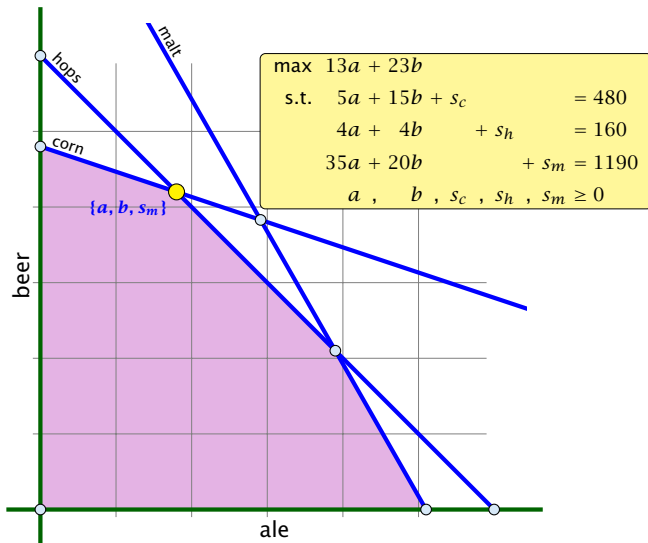
Note that with this interpretation, complementary slackness becomes obvious.

- ▶ If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- ▶ If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

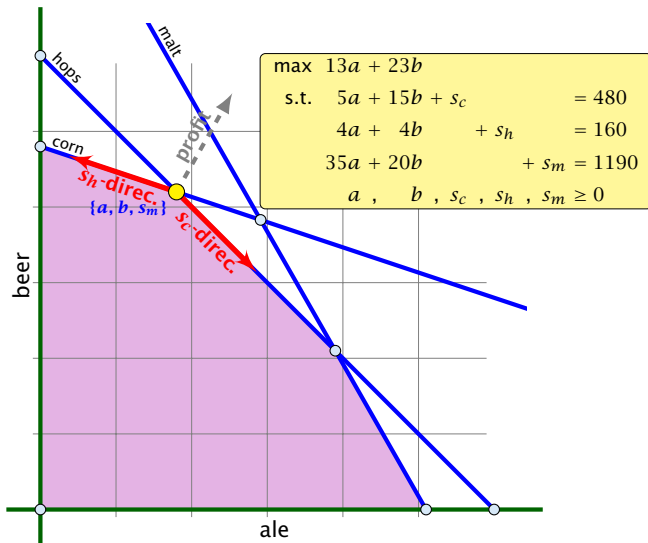
Example



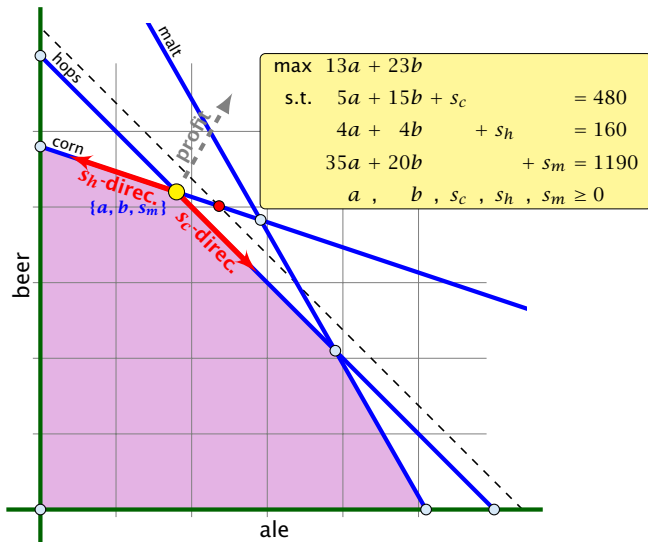
Example



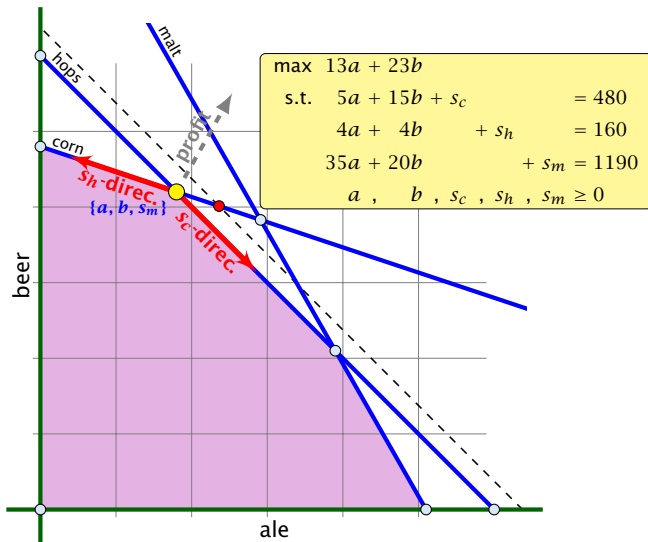
Example



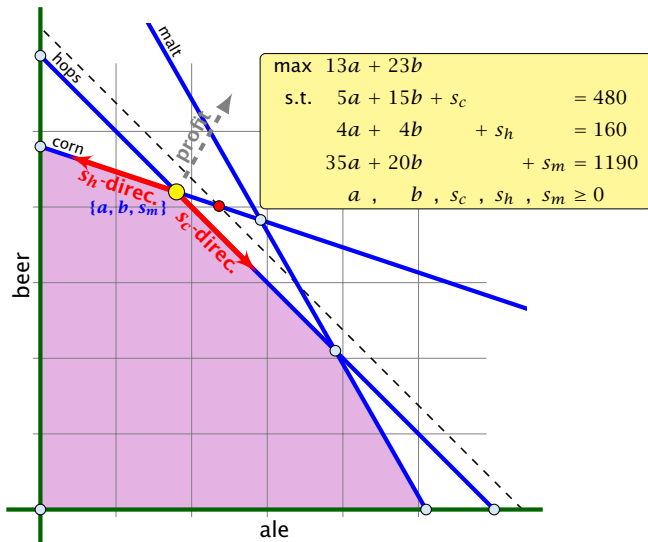
Example



Example



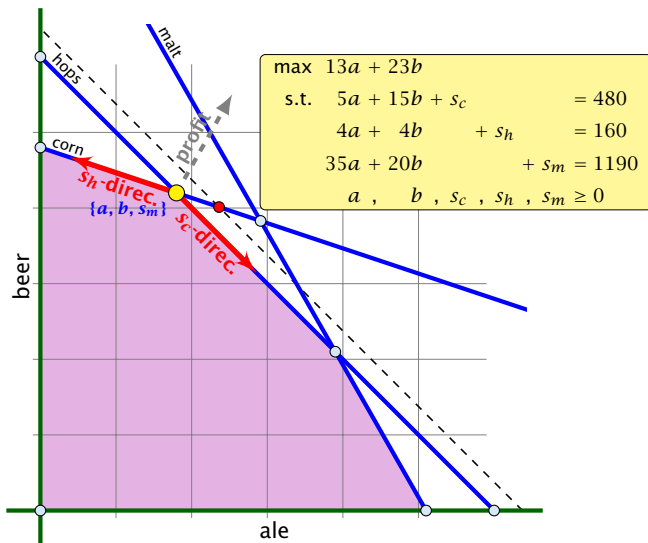
Example



The change in profit when increasing hops by one unit is

$$= c_B^T A_B^{-1} e_h.$$

Example



The change in profit when increasing hops by one unit is

$$= \underbrace{c_B^T A_B^{-1}}_{y^*} e_h.$$

Of course, the previous argument about the increase in the primal objective only holds for the non-degenerate case.

If the optimum basis is degenerate then increasing the supply of one resource may not allow the objective value to increase.

Definition 42

An (s, t) -flow in a (complete) directed graph $G = (V, V \times V, c)$ is a function $f : V \times V \mapsto \mathbb{R}_0^+$ that satisfies

1. For each edge (x, y)

$$0 \leq f_{xy} \leq c_{xy} .$$

(capacity constraints)

2. For each $v \in V \setminus \{s, t\}$

$$\sum_x f_{vx} = \sum_x f_{xv} .$$

(flow conservation constraints)

Flows

Definition 42

An (s, t) -flow in a (complete) directed graph $G = (V, V \times V, c)$ is a function $f : V \times V \mapsto \mathbb{R}_0^+$ that satisfies

1. For each edge (x, y)

$$0 \leq f_{xy} \leq c_{xy} .$$

(capacity constraints)

2. For each $v \in V \setminus \{s, t\}$

$$\sum_x f_{vx} = \sum_x f_{xv} .$$

(flow conservation constraints)

Definition 43

The **value of an (s, t) -flow f** is defined as

$$\text{val}(f) = \sum_x f_{sx} - \sum_x f_{xs} .$$

Maximum Flow Problem:

Find an (s, t) -flow with maximum value.

Definition 43

The **value of an (s, t) -flow f** is defined as

$$\text{val}(f) = \sum_x f_{sx} - \sum_x f_{xs} .$$

Maximum Flow Problem:

Find an (s, t) -flow with maximum value.

LP-Formulation of Maxflow

$$\begin{array}{ll}\max & \sum_z f_{sz} - \sum_z f_{zs} \\ \text{s.t.} & \forall (z, w) \in V \times V \quad f_{zw} \leq c_{zw} \quad \ell_{zw} \\ & \forall w \neq s, t \quad \sum_z f_{zw} - \sum_z f_{wz} = 0 \quad p_w \\ & f_{zw} \geq 0\end{array}$$

LP-Formulation of Maxflow

$$\begin{array}{ll}
 \max & \sum_z f_{sz} - \sum_z f_{zs} \\
 \text{s.t.} & \forall (z, w) \in V \times V \quad f_{zw} \leq c_{zw} \quad \ell_{zw} \\
 & \forall w \neq s, t \quad \sum_z f_{zw} - \sum_z f_{wz} = 0 \quad p_w \\
 & f_{zw} \geq 0
 \end{array}$$

$$\begin{array}{ll}
 \min & \sum_{(xy)} c_{xy} \ell_{xy} \\
 \text{s.t.} & f_{xy} \ (x, y \neq s, t) : \quad 1\ell_{xy} - 1p_x + 1p_y \geq 0 \\
 & f_{sy} \ (y \neq s, t) : \quad 1\ell_{sy} \quad \quad + 1p_y \geq 1 \\
 & f_{xs} \ (x \neq s, t) : \quad 1\ell_{xs} - 1p_x \quad \quad \geq -1 \\
 & f_{ty} \ (y \neq s, t) : \quad 1\ell_{ty} \quad \quad + 1p_y \geq 0 \\
 & f_{xt} \ (x \neq s, t) : \quad 1\ell_{xt} - 1p_x \quad \quad \geq 0 \\
 & f_{st} : \quad \quad \quad 1\ell_{st} \quad \quad \geq 1 \\
 & f_{ts} : \quad \quad \quad 1\ell_{ts} \quad \quad \geq -1 \\
 & \quad \quad \quad \ell_{xy} \quad \quad \geq 0
 \end{array}$$

LP-Formulation of Maxflow

$$\begin{array}{ll} \min & \sum_{(x,y)} c_{xy} \ell_{xy} \\ \text{s.t.} & f_{xy} \ (x, y \neq s, t) : \quad 1\ell_{xy} - 1p_x + 1p_y \geq 0 \\ & f_{sy} \ (y \neq s, t) : \quad 1\ell_{sy} - \quad 1 + 1p_y \geq 0 \\ & f_{xs} \ (x \neq s, t) : \quad 1\ell_{xs} - 1p_x + \quad 1 \geq 0 \\ & f_{ty} \ (y \neq s, t) : \quad 1\ell_{ty} - \quad 0 + 1p_y \geq 0 \\ & f_{xt} \ (x \neq s, t) : \quad 1\ell_{xt} - 1p_x + \quad 0 \geq 0 \\ & f_{st} : \quad 1\ell_{st} - \quad 1 + \quad 0 \geq 0 \\ & f_{ts} : \quad 1\ell_{ts} - \quad 0 + \quad 1 \geq 0 \\ & \ell_{xy} \geq 0 \end{array}$$

LP-Formulation of Maxflow

$$\begin{array}{ll} \min & \sum_{(x,y)} c_{xy} \ell_{xy} \\ \text{s.t.} & f_{xy} \ (x, y \neq s, t) : \quad 1\ell_{xy} - 1p_x + 1p_y \geq 0 \\ & f_{sy} \ (y \neq s, t) : \quad 1\ell_{sy} - p_s + 1p_y \geq 0 \\ & f_{xs} \ (x \neq s, t) : \quad 1\ell_{xs} - 1p_x + p_s \geq 0 \\ & f_{ty} \ (y \neq s, t) : \quad 1\ell_{ty} - p_t + 1p_y \geq 0 \\ & f_{xt} \ (x \neq s, t) : \quad 1\ell_{xt} - 1p_x + p_t \geq 0 \\ & f_{st} : \quad 1\ell_{st} - p_s + p_t \geq 0 \\ & f_{ts} : \quad 1\ell_{ts} - p_t + p_s \geq 0 \\ & \ell_{xy} \geq 0 \end{array}$$

with $p_t = 0$ and $p_s = 1$.

LP-Formulation of Maxflow

$$\begin{array}{ll}\min & \sum_{(x,y)} c_{xy} \ell_{xy} \\ \text{s.t.} & f_{xy} : \quad 1\ell_{xy} - 1p_x + 1p_y \geq 0 \\ & \ell_{xy} \geq 0 \\ & p_s = 1 \\ & p_t = 0\end{array}$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t)$).

LP-Formulation of Maxflow

$$\begin{array}{ll}\min & \sum_{(x,y)} c_{xy} \ell_{xy} \\ \text{s.t. } f_{xy} : & \ell_{xy} - p_x + p_y \geq 0 \\ & \ell_{xy} \geq 0 \\ & p_s = 1 \\ & p_t = 0\end{array}$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t)$).

LP-Formulation of Maxflow

$$\begin{array}{ll}\min & \sum_{(xy)} c_{xy} \ell_{xy} \\ \text{s.t.} & f_{xy} : \quad 1\ell_{xy} - 1p_x + 1p_y \geq 0 \\ & \ell_{xy} \geq 0 \\ & p_s = 1 \\ & p_t = 0\end{array}$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x, t) \leq d(x, y) + d(y, t) \Rightarrow d(x, t) \leq \ell_{xy} + d(y, t)$).

LP-Formulation of Maxflow

$$\begin{array}{ll}\min & \sum_{(xy)} c_{xy} \ell_{xy} \\ \text{s.t.} & f_{xy} : \quad 1\ell_{xy} - 1p_x + 1p_y \geq 0 \\ & \ell_{xy} \geq 0 \\ & p_s = 1 \\ & p_t = 0\end{array}$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x, t) \leq d(x, y) + d(y, t) \Rightarrow d(x, t) \leq \ell_{xy} + d(y, t)$).

One can show that there is an optimum LP-solution for the dual problem that gives an integral assignment of variables.

This means $p_x = 1$ or $p_x = 0$ for our case. This gives rise to a cut in the graph with vertices having value 1 on one side and the other vertices on the other side. The objective function then evaluates the capacity of this cut.

This shows that the Maxflow/Mincut theorem follows from linear programming duality.

One can show that there is an optimum LP-solution for the dual problem that gives an integral assignment of variables.

This means $p_x = 1$ or $p_x = 0$ for our case. This gives rise to a cut in the graph with vertices having value 1 on one side and the other vertices on the other side. The objective function then evaluates the capacity of this cut.

This shows that the Maxflow/Mincut theorem follows from linear programming duality.

One can show that there is an optimum LP-solution for the dual problem that gives an integral assignment of variables.

This means $p_x = 1$ or $p_x = 0$ for our case. This gives rise to a cut in the graph with vertices having value 1 on one side and the other vertices on the other side. The objective function then evaluates the capacity of this cut.

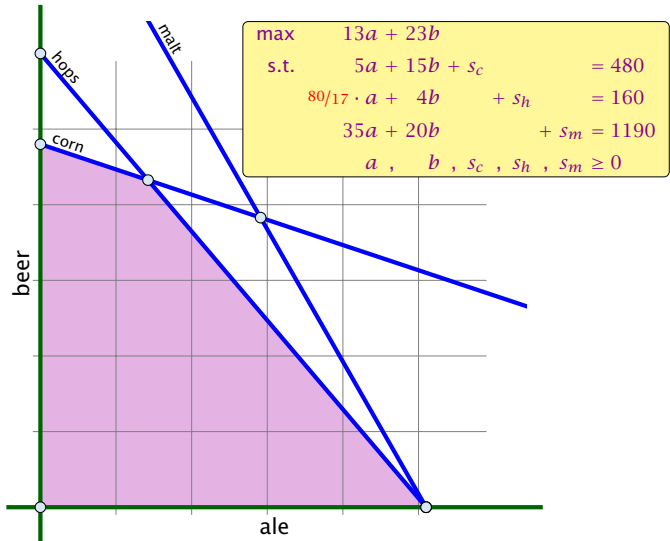
This shows that the Maxflow/Mincut theorem follows from linear programming duality.

Degeneracy Revisited

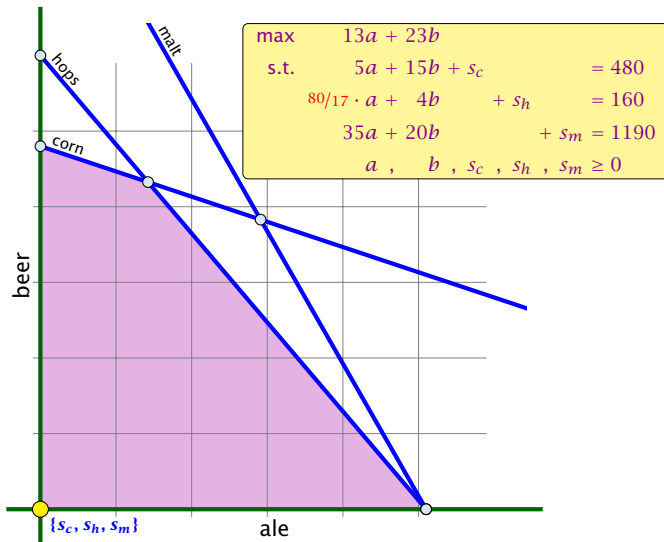
Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

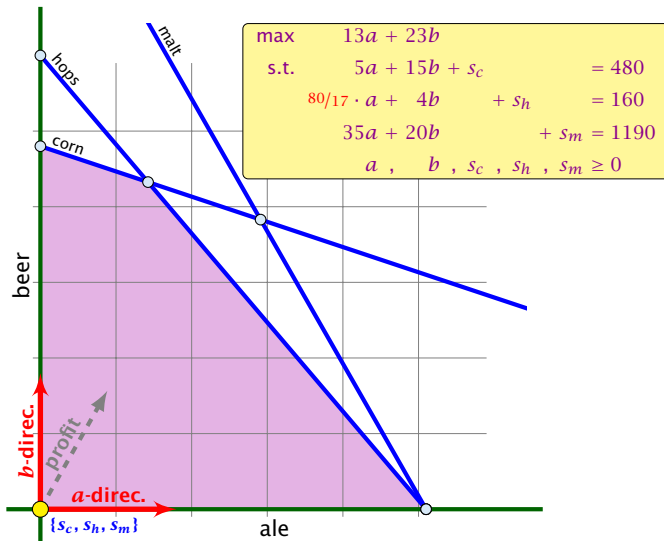
Degenerate Example



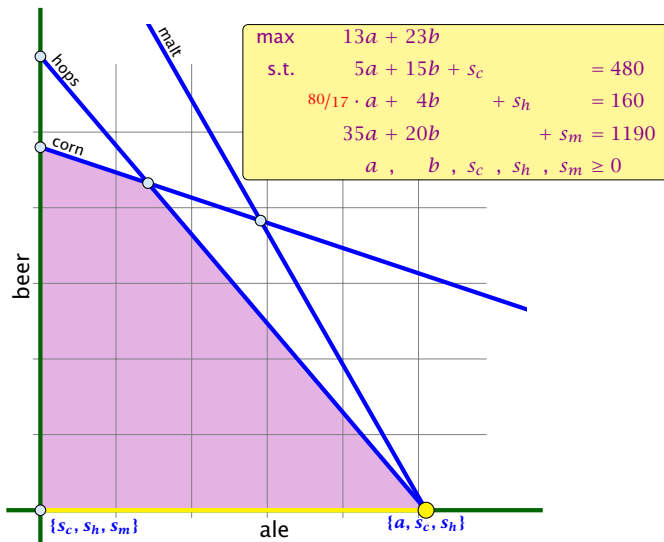
Degenerate Example



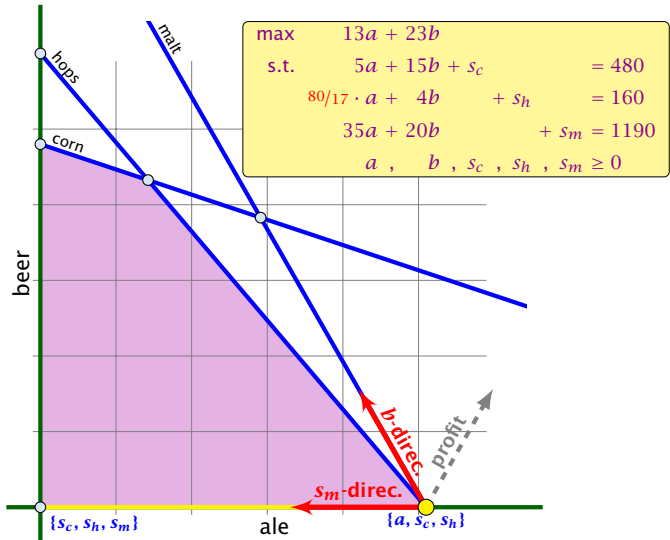
Degenerate Example



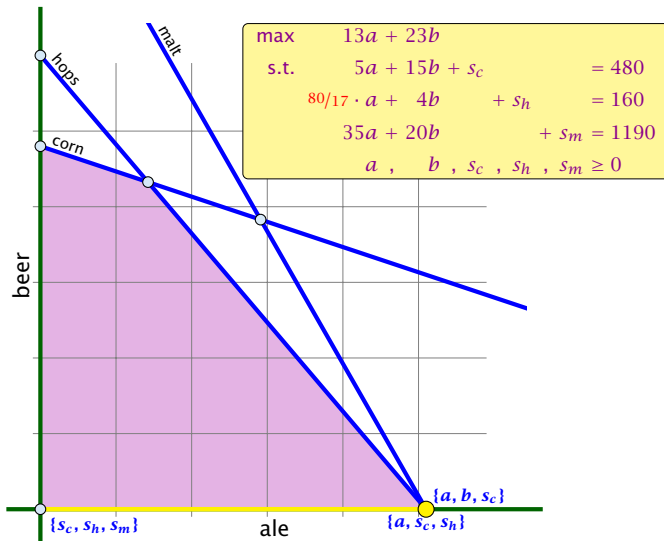
Degenerate Example



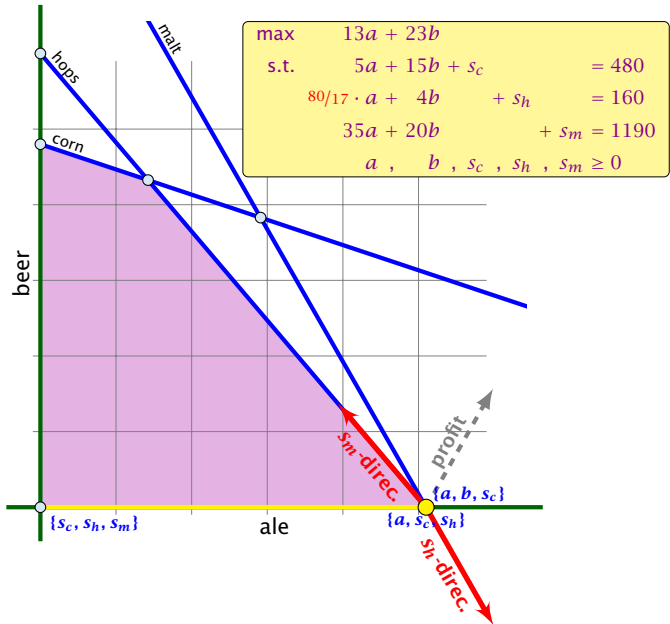
Degenerate Example



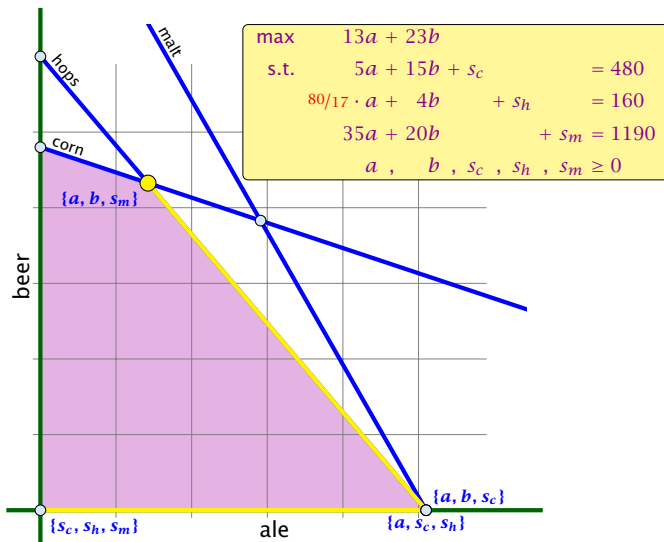
Degenerate Example



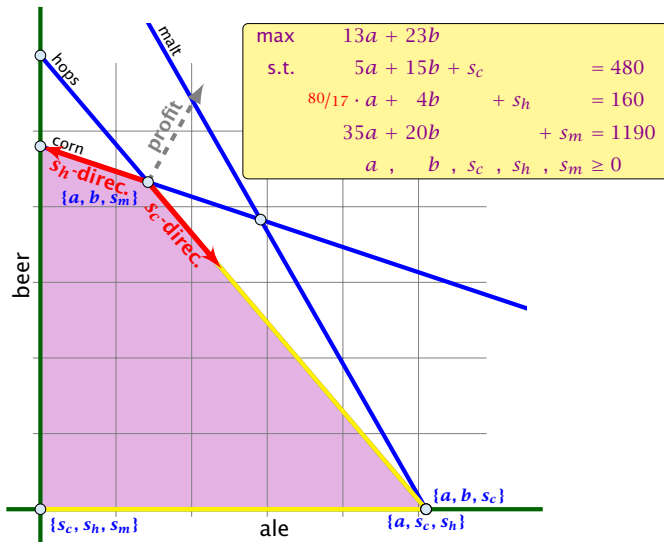
Degenerate Example



Degenerate Example



Degenerate Example



Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:

Given feasible $LP := \max\{c^T x, Ax = b; x \geq 0\}$. Change it into $LP' := \max\{c^T x, Ax = b', x \geq 0\}$ such that

1. LP' is feasible

- 2. If a set of basis variables corresponds to an optimal basis in LP' (e.g., B') then it corresponds to an infeasible basis in LP (e.g., B) where the columns in B are linearly independent, but b has no degenerate basic solution.

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:

Given feasible $LP := \max\{c^T x, Ax = b; x \geq 0\}$. Change it into $LP' := \max\{c^T x, Ax = b', x \geq 0\}$ such that

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:

Given feasible $LP := \max\{c^T x, Ax = b; x \geq 0\}$. Change it into $LP' := \max\{c^T x, Ax = b', x \geq 0\}$ such that

- I. LP' is feasible
- II. If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1}b \not\geq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).
- III. LP' has no degenerate basic solutions

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:

Given feasible $LP := \max\{c^T x, Ax = b; x \geq 0\}$. Change it into $LP' := \max\{c^T x, Ax = b', x \geq 0\}$ such that

- I. LP' is feasible
- II. If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1}b \not\geq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).
- III. LP' has no degenerate basic solutions

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:

Given feasible $LP := \max\{c^T x, Ax = b; x \geq 0\}$. Change it into $LP' := \max\{c^T x, Ax = b', x \geq 0\}$ such that

- I. LP' is feasible
- II. If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1}b \not\geq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).
- III. LP' has no degenerate basic solutions

Perturbation

Let B be index set of **some** basis with basic solution

$$x_B^* = A_B^{-1}b \geq 0, x_N^* = 0 \quad (\text{i.e. } B \text{ is feasible})$$

Fix

$$b' := b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \quad \text{for } \varepsilon > 0 .$$

This is the perturbation that we are using.

Perturbation

Let B be index set of **some** basis with basic solution

$$x_B^* = A_B^{-1}b \geq 0, x_N^* = 0 \quad (\text{i.e. } B \text{ is feasible})$$

Fix

$$b' := b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \quad \text{for } \varepsilon > 0 .$$

This is the perturbation that we are using.

Property I

The new LP is feasible because the set B of basis variables provides a feasible basis:

$$A_B^{-1} \left(b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right) = x_B^* + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \geq 0 \ .$$

Property I

The new LP is feasible because the set B of basis variables provides a feasible basis:

$$A_B^{-1} \left(b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right) = x_B^* + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \geq 0 .$$

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i .

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i .

Then for small enough $\epsilon > 0$

$$\left(A_{\tilde{B}}^{-1} \left(b + A_B \begin{pmatrix} \epsilon \\ \vdots \\ \epsilon^m \end{pmatrix} \right) \right)_i$$

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i .

Then for small enough $\epsilon > 0$

$$\left(A_{\tilde{B}}^{-1} \left(b + A_B \begin{pmatrix} \epsilon \\ \vdots \\ \epsilon^m \end{pmatrix} \right) \right)_i = (A_{\tilde{B}}^{-1}b)_i + \left(A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \epsilon \\ \vdots \\ \epsilon^m \end{pmatrix} \right)_i < 0$$

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i .

Then for small enough $\epsilon > 0$

$$\left(A_{\tilde{B}}^{-1} \left(b + A_B \begin{pmatrix} \epsilon \\ \vdots \\ \epsilon^m \end{pmatrix} \right) \right)_i = (A_{\tilde{B}}^{-1}b)_i + \left(A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \epsilon \\ \vdots \\ \epsilon^m \end{pmatrix} \right)_i < 0$$

Hence, $\tilde{B}$ is not feasible.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x_{\tilde{B}}^* = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynomial with variable ε of degree at most m .

$A_{\tilde{B}}^{-1}A_B$ has rank m . Therefore no polynomial is 0.

A polynomial of degree at most m has at most m roots (Nullstellen).

Hence, $\varepsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x_{\tilde{B}}^* = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynomial with variable ε of degree at most m .

$A_{\tilde{B}}^{-1}A_B$ has rank m . Therefore no polynomial is 0.

A polynomial of degree at most m has at most m roots (Nullstellen).

Hence, $\varepsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x_{\tilde{B}}^* = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynomial with variable ε of degree at most m .

$A_{\tilde{B}}^{-1}A_B$ has rank m . Therefore no polynomial is 0.

A polynomial of degree at most m has at most m roots (Nullstellen).

Hence, $\varepsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x_{\tilde{B}}^* = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynomial with variable ε of degree at most m .

$A_{\tilde{B}}^{-1}A_B$ has rank m . Therefore no polynomial is 0.

A polynomial of degree at most m has at most m roots (Nullstellen).

Hence, $\varepsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x_{\tilde{B}}^* = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynomial with variable ε of degree at most m .

$A_{\tilde{B}}^{-1}A_B$ has rank m . Therefore no polynomial is 0.

A polynomial of degree at most m has at most m roots (Nullstellen).

Hence, $\varepsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x_{\tilde{B}}^* = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynomial with variable ε of degree at most m .

$A_{\tilde{B}}^{-1}A_B$ has rank m . Therefore no polynomial is 0.

A polynomial of degree at most m has at most m roots (Nullstellen).

Hence, $\varepsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Since, there are no degeneracies Simplex will terminate when run on LP' .

Since, there are no degeneracies Simplex will terminate when run on LP' .

- ▶ If it terminates because the reduced cost vector fulfills

$$\tilde{c} = (c^T - c_B^T A_B^{-1} A) \leq 0$$

then we have found an optimal basis.

Since, there are no degeneracies Simplex will terminate when run on LP' .

- ▶ If it terminates because the reduced cost vector fulfills

$$\tilde{c} = (c^T - c_B^T A_B^{-1} A) \leq 0$$

then we have found an optimal basis. **Note that this basis is also optimal for LP , as the above constraint does not depend on b .**

Since, there are no degeneracies Simplex will terminate when run on LP' .

- ▶ If it terminates because the reduced cost vector fulfills

$$\tilde{c} = (c^T - c_B^T A_B^{-1} A) \leq 0$$

then we have found an optimal basis. **Note that this basis is also optimal for LP , as the above constraint does not depend on b .**

- ▶ If it terminates because it finds a variable x_j with $\tilde{c}_j > 0$ for which the j -th basis direction d , fulfills $d \geq 0$ we know that LP' is unbounded. The basis direction **does not depend on b** . Hence, we also know that LP is unbounded.

Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of ε is difficult.

Idea:

Simulate behaviour of LP' without explicitly doing a perturbation.

Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of ε is difficult.

Idea:

Simulate behaviour of LP' without explicitly doing a perturbation.

Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of ε is difficult.

Idea:

Simulate behaviour of LP' without explicitly doing a perturbation.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

In the following we assume that $b \geq 0$. This can be obtained by replacing the initial system $(A \mid b)$ by $(A_B^{-1}A \mid A_B^{-1}b)$ where B is the index set of a feasible basis (found e.g. by the first phase of the Two-phase algorithm).

Then the perturbed instance is

$$b' = b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

Lexicographic Pivoting

In the following we assume that $b \geq 0$. This can be obtained by replacing the initial system $(A \mid b)$ by $(A_B^{-1}A \mid A_B^{-1}b)$ where B is the index set of a feasible basis (found e.g. by the first phase of the Two-phase algorithm).

Then the perturbed instance is

$$b' = b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

Matrix View

Let our linear program be

$$\begin{aligned}c_B^T x_B + c_N^T x_N &= Z \\ A_B x_B + A_N x_N &= b \\ x_B, x_N &\geq 0\end{aligned}$$

The simplex tableaux for basis B is

$$\begin{aligned}(c_N^T - c_B^T A_B^{-1} A_N) x_N &= Z - c_B^T A_B^{-1} b \\ I x_B + A_B^{-1} A_N x_N &= A_B^{-1} b \\ x_B, x_N &\geq 0\end{aligned}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^T - c_B^T A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_{\ell} = \frac{\hat{b}_{\ell}}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_{\ell}}{(A_B^{-1}A_{*e})_{\ell}} .$$

ℓ is the index of a leaving variable within B . This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_{\ell} = \frac{\hat{b}_{\ell}}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_{\ell}}{(A_B^{-1}A_{*e})_{\ell}} .$$

ℓ is the index of a leaving variable within B . This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_{\ell} = \frac{\hat{b}_{\ell}}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_{\ell}}{(A_B^{-1}A_{*e})_{\ell}} .$$

ℓ is the index of a leaving variable within B . This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_{\ell} = \frac{\hat{b}_{\ell}}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_{\ell}}{(A_B^{-1}A_{*e})_{\ell}} .$$

ℓ is the index of a leaving variable within B . This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

Definition 44

$u \leq_{\text{lex}} v$ if and only if the first component in which u and v differ fulfills $u_i \leq v_i$.

Lexicographic Pivoting

LP' chooses an index that minimizes

$$\theta_\ell$$

Lexicographic Pivoting

LP' chooses an index that minimizes

$$\theta_\ell = \frac{\left(A_B^{-1} \left(b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right) \right)_\ell}{(A_B^{-1} A_{*e})_\ell}$$

Lexicographic Pivoting

LP' chooses an index that minimizes

$$\theta_\ell = \frac{\left(A_B^{-1} \left(b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right) \right)_\ell}{(A_B^{-1} A_{*e})_\ell} = \frac{\left(A_B^{-1} (b \mid I) \begin{pmatrix} 1 \\ \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right)_\ell}{(A_B^{-1} A_{*e})_\ell}$$

Lexicographic Pivoting

LP' chooses an index that minimizes

$$\begin{aligned}\theta_\ell &= \frac{\left(A_B^{-1} \left(b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right) \right)_\ell}{(A_B^{-1} A_{*e})_\ell} = \frac{\left(A_B^{-1}(b \mid I) \begin{pmatrix} 1 \\ \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \right)_\ell}{(A_B^{-1} A_{*e})_\ell} \\ &= \frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1} A_{*e})_\ell} \begin{pmatrix} 1 \\ \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}\end{aligned}$$

Lexicographic Pivoting

This means you can choose the variable/row ℓ for which the vector

$$\frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell > 0$.

This technique guarantees that your pivoting is the same as in the perturbed case. This guarantees that cycling does not occur.

Lexicographic Pivoting

This means you can choose the variable/row ℓ for which the vector

$$\frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell > 0$.

This technique guarantees that your pivoting is the same as in the perturbed case. This guarantees that cycling does not occur.

Lexicographic Pivoting

This means you can choose the variable/row ℓ for which the vector

$$\frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell > 0$.

This technique guarantees that your pivoting is the same as in the perturbed case. This guarantees that cycling does not occur.

Number of Simplex Iterations

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

The input size is $L \cdot n \cdot m$, where n is the number of variables, m is the number of constraints, and L is the length of the binary representation of the largest coefficient in the matrix A .

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

The input size is $L \cdot n \cdot m$, where n is the number of variables, m is the number of constraints, and L is the length of the binary representation of the largest coefficient in the matrix A .

If we really require $\binom{n}{m}$ iterations then Simplex is not a polynomial time algorithm.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

The input size is $L \cdot n \cdot m$, where n is the number of variables, m is the number of constraints, and L is the length of the binary representation of the largest coefficient in the matrix A .

If we really require $\binom{n}{m}$ iterations then Simplex is not a polynomial time algorithm.

Can we obtain a better analysis?

Number of Simplex Iterations

Observation

Simplex visits every **feasible** basis at most once.

Number of Simplex Iterations

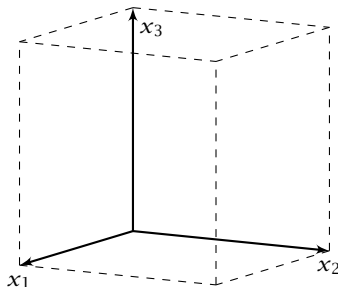
Observation

Simplex visits every **feasible** basis at most once.

However, also the number of feasible bases can be very large.

Example

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & 0 \leq x_1 \leq 1 \\ & 0 \leq x_2 \leq 1 \\ & \vdots \\ & 0 \leq x_n \leq 1\end{array}$$

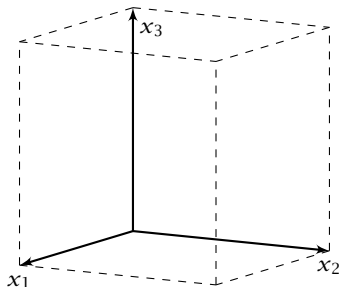


$2n$ constraint on n variables define an n -dimensional hypercube as feasible region.

The feasible region has 2^n vertices.

Example

$$\begin{array}{ll}\max & c^T x \\ \text{s.t.} & 0 \leq x_1 \leq 1 \\ & 0 \leq x_2 \leq 1 \\ & \vdots \\ & 0 \leq x_n \leq 1\end{array}$$



However, Simplex may still run quickly as it usually does not visit all feasible bases.

In the following we give an example of a feasible region for which there is a bad **Pivoting Rule**.

Pivoting Rule

A Pivoting Rule defines how to choose the entering and leaving variable for an iteration of Simplex.

In the non-degenerate case after choosing the entering variable the leaving variable is unique.

Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

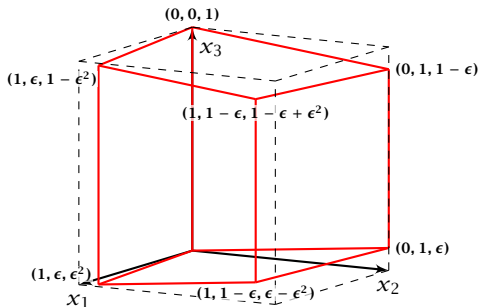
$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$

$$\vdots$$

$$\epsilon x_{n-1} \leq x_n \leq 1 - \epsilon x_{n-1}$$

$$x_i \geq 0$$



Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

Observations

- ▶ We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- ▶ Every basis is defined by $2n$ variables, and n non-basic variables.
- ▶ There exist degenerate vertices.
- ▶ The degeneracies come from the non-negativity constraints, which are superfluous.
- ▶ In the following all variables x_i stay in the basis at all times.
- ▶ Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- ▶ We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \rightarrow 0$.

- ▶ In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- ▶ The basis $(0, \dots, 0, 1)$ is the unique optimal basis.
- ▶ Our sequence S_n starts at $(0, \dots, 0)$ ends with $(0, \dots, 0, 1)$ and visits every node of the hypercube.
- ▶ An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

Analysis

- ▶ In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- ▶ The basis $(0, \dots, 0, 1)$ is the unique optimal basis.
- ▶ Our sequence S_n starts at $(0, \dots, 0)$ ends with $(0, \dots, 0, 1)$ and visits every node of the hypercube.
- ▶ An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

Analysis

- ▶ In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- ▶ The basis $(0, \dots, 0, 1)$ is the unique optimal basis.
- ▶ Our sequence S_n starts at $(0, \dots, 0)$ ends with $(0, \dots, 0, 1)$ and visits every node of the hypercube.
- ▶ An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

- ▶ In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- ▶ The basis $(0, \dots, 0, 1)$ is the unique optimal basis.
- ▶ Our sequence S_n starts at $(0, \dots, 0)$ ends with $(0, \dots, 0, 1)$ and visits every node of the hypercube.
- ▶ An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

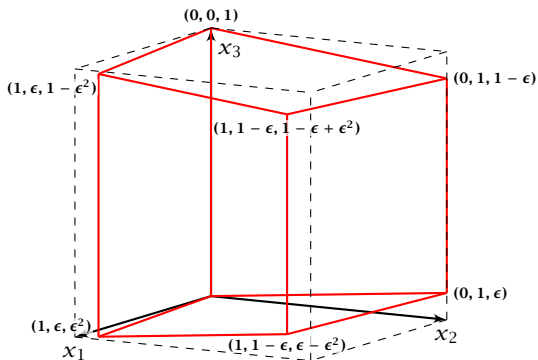
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



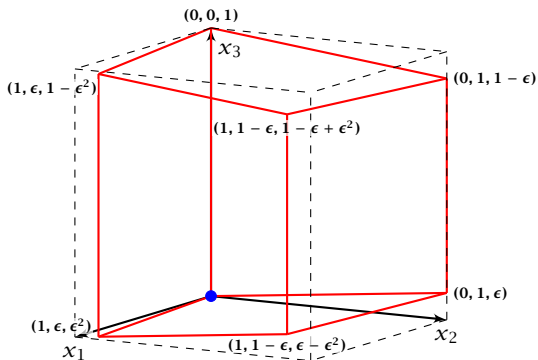
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



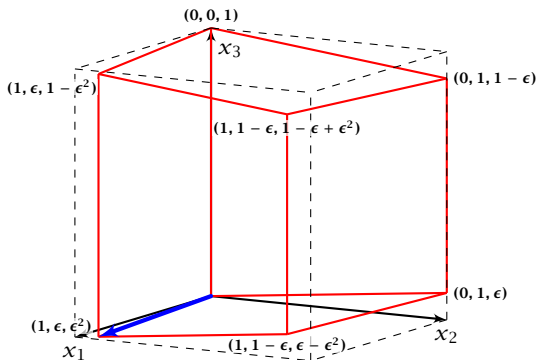
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



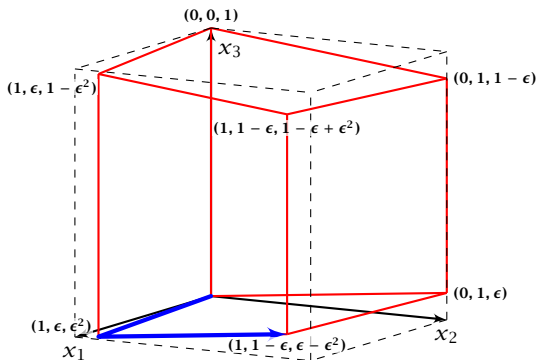
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



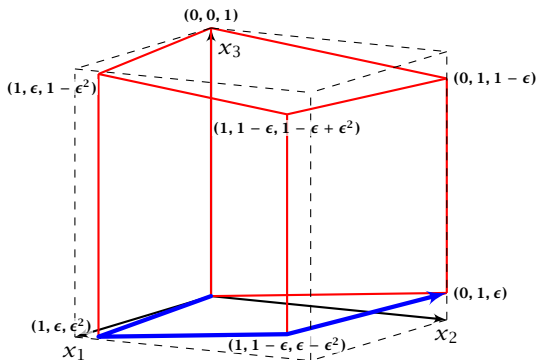
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



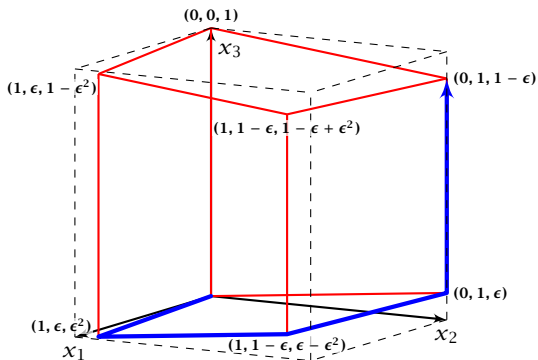
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



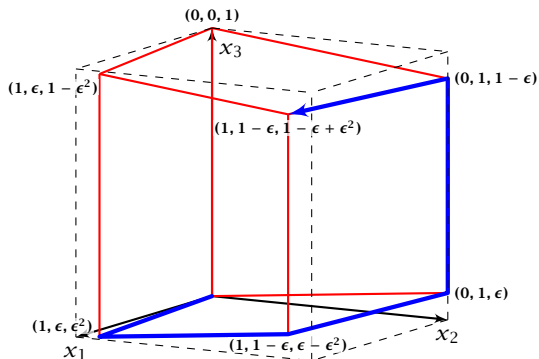
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



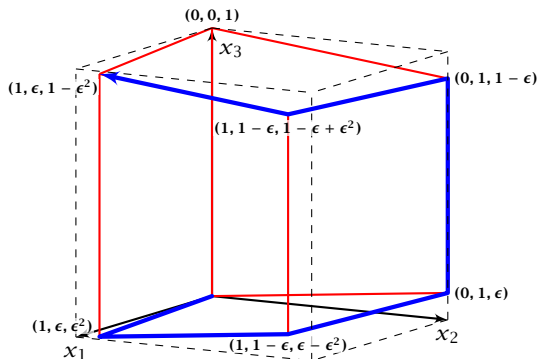
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



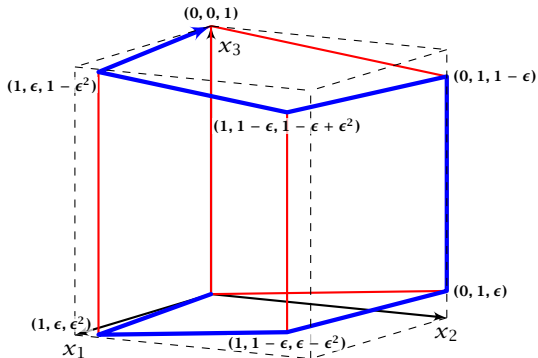
Klee Minty Cube

$$\max x_n$$

$$\text{s.t.} \quad 0 \leq x_1 \leq 1$$

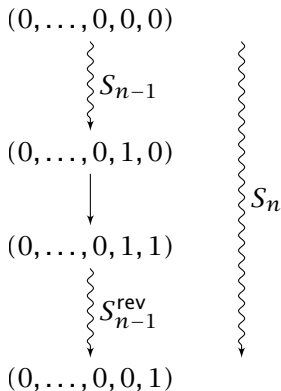
$$\epsilon x_1 \leq x_2 \leq 1 - \epsilon x_1$$

$$\epsilon x_2 \leq x_3 \leq 1 - \epsilon x_2$$



Analysis

The sequence S_n that visits every node of the hypercube is defined recursively



The non-recursive case is $S_1 = 0 \rightarrow 1$

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

For the first part the value of x_{n-1} is increasing along S_{n-1} .

By induction hypothesis x_{n-1} is increasing along S_{n-1} .
By choice, also

Going from S_{n-1} to S_n , x_{n-1} decreases. If for ϵ small enough $x_{n-1} > 0$, then x_{n-1} is increasing along S_n .

For the remaining path S_n we have $x_{n-1} = 0$ and

By induction hypothesis x_n is increasing along S_n , hence
It is increasing along S_n .

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

For the first part the value of x_n is increasing along S_n because of the induction hypothesis. x_{n-1} is increasing along S_{n-1} twice, also. Going from S_{n-1} to S_n , x_{n-1} decreases a bit, but for ϵ small enough x_{n-1} is still increasing along S_n . For the remaining path S_n we have $x_{n-1} = 0$ and x_n is increasing along S_n because of the induction hypothesis. x_n is increasing along S_n , hence x_n is increasing along S_n .

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

At the first part the value of x_n is increasing, since x_n is increasing by induction hypothesis, and x_n is increasing along S_{n-1} as well, also. Then, x_n is decreasing along S_{n-1} for n large enough, since x_n is decreasing along S_{n-1} for n large enough. For the remaining part of the path, we have x_n is increasing along S_{n-1} for n large enough, and x_n is increasing along S_{n-1} for n large enough. If increasing along S_{n-1} for n large enough, then x_n is increasing along S_{n-1} for n large enough.

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

- ▶ For the first part the value of $x_n = \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence, also x_n .
- ▶ Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ .
- ▶ For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev} .

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

- ▶ For the first part the value of $x_n = \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence, also x_n .
- ▶ Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ .
- ▶ For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev} .

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

- ▶ For the first part the value of $x_n = \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence, also x_n .
- ▶ Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ .
- ▶ For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev} .

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

- ▶ For the first part the value of $x_n = \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence, also x_n .
- ▶ Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ .
- ▶ For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
 - ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev} .

Analysis

Lemma 45

The objective value x_n is increasing along path S_n .

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$n - 1 \rightarrow n$

- ▶ For the first part the value of $x_n = \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence, also x_n .
- ▶ Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ .
- ▶ For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- ▶ By induction hypothesis x_{n-1} is increasing along S_{n-1} , hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev} .

Remarks about Simplex

Observation

The simplex algorithm takes at most $\binom{n}{m}$ iterations. Each iteration can be implemented in time $\mathcal{O}(mn)$.

In practise it usually takes a linear number of iterations.

Remarks about Simplex

Theorem

For almost all known **deterministic** pivoting rules (rules for choosing entering and leaving variables) there exist lower bounds that require the algorithm to have exponential running time ($\Omega(2^{\Omega(n)})$) (e.g. Klee Minty 1972).

Remarks about Simplex

Theorem

For some standard **randomized** pivoting rules there exist subexponential lower bounds ($\Omega(2^{\Omega(n^\alpha)})$ for $\alpha > 0$) (Friedmann, Hansen, Zwick 2011).

Remarks about Simplex

Conjecture (Hirsch 1957)

The edge-vertex graph of an m -facet polytope in d -dimensional Euclidean space has diameter no more than $m - d$.

The conjecture has been proven wrong in 2010.

But the question whether the diameter is perhaps of the form $\mathcal{O}(\text{poly}(m, d))$ is open.

8 Seidels LP-algorithm

- ▶ Suppose we want to solve $\min\{c^T x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- ▶ In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- ▶ If d is much smaller than m one can do a lot better.
- ▶ In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m .

8 Seidels LP-algorithm

- ▶ Suppose we want to solve $\min\{c^T x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- ▶ In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- ▶ If d is much smaller than m one can do a lot better.
- ▶ In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m .

8 Seidels LP-algorithm

- ▶ Suppose we want to solve $\min\{c^T x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- ▶ In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- ▶ If d is much smaller than m one can do a lot better.
- ▶ In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m .

8 Seidels LP-algorithm

- ▶ Suppose we want to solve $\min\{c^T x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- ▶ In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- ▶ If d is much smaller than m one can do a lot better.
- ▶ In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., **linear in m** .

8 Seidels LP-algorithm

Setting:

- ▶ We assume an LP of the form

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0\end{array}$$

- ▶ We assume that the LP is **bounded**.

Ensuring Conditions

Given a **standard minimization LP**

$$\begin{array}{ll}\min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0\end{array}$$

how can we obtain an LP of the required form?

- ▶ **Compute a lower bound on $c^T x$ for any basic feasible solution.**

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b .

Multiply entries in A, b by s to obtain integral entries. This does not change the feasible region.

Add slack variables to A ; denote the resulting matrix with $\tilde{A}$.

If B is an optimal basis then x_B with $\tilde{A}_B x_B = \tilde{b}$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b .

Multiply entries in A, b by s to obtain integral entries. **This does not change the feasible region.**

Add slack variables to A ; denote the resulting matrix with $\tilde{A}$.

If B is an optimal basis then x_B with $\tilde{A}_B x_B = \tilde{b}$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b .

Multiply entries in A, b by s to obtain integral entries. **This does not change the feasible region.**

Add slack variables to A ; denote the resulting matrix with $\bar{A}$.

If B is an optimal basis then x_B with $\bar{A}_B x_B = \bar{b}$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b .

Multiply entries in A, b by s to obtain integral entries. **This does not change the feasible region.**

Add slack variables to A ; denote the resulting matrix with $\bar{A}$.

If B is an optimal basis then x_B with $\bar{A}_B x_B = \bar{b}$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Theorem 46 (Cramers Rule)

Let M be a matrix with $\det(M) \neq 0$. Then the solution to the system $Mx = b$ is given by

$$x_i = \frac{\det(M_i)}{\det(M)} ,$$

where M_i is the matrix obtained from M by replacing the i -th column by the vector b .

Proof:

Proof:

- ▶ Define

$$X_i = \begin{pmatrix} | & & | & | & | & & | \\ e_1 & \cdots & e_{i-1} & x & e_{i+1} & \cdots & e_n \\ | & & | & | & | & & | \end{pmatrix}$$

Note that expanding along the i -th column gives that $\det(X_i) = x_i$.

- ▶ Further, we have

$$MX_i = \begin{pmatrix} | & & | & | & | & & | \\ Me_1 & \cdots & Me_{i-1} & Mx & Me_{i+1} & \cdots & Me_n \\ | & & | & | & | & & | \end{pmatrix} = M_i$$

- ▶ Hence,

$$x_i = \det(X_i) = \frac{\det(M_i)}{\det(M)}$$

Proof:

- ▶ Define

$$X_i = \begin{pmatrix} | & & | & | & | & & | \\ e_1 & \cdots & e_{i-1} & x & e_{i+1} & \cdots & e_n \\ | & & | & | & | & & | \end{pmatrix}$$

Note that expanding along the i -th column gives that $\det(X_i) = x_i$.

- ▶ Further, we have

$$MX_i = \begin{pmatrix} | & & | & | & | & & | \\ Me_1 & \cdots & Me_{i-1} & Mx & Me_{i+1} & \cdots & Me_n \\ | & & | & | & | & & | \end{pmatrix} = M_i$$

- ▶ Hence,

$$x_i = \det(X_i) = \frac{\det(M_i)}{\det(M)}$$

Proof:

- ▶ Define

$$X_i = \begin{pmatrix} | & & | & | & | & \\ e_1 & \cdots & e_{i-1} & x & e_{i+1} & \cdots & e_n \\ | & & | & | & | & \end{pmatrix}$$

Note that expanding along the i -th column gives that $\det(X_i) = x_i$.

- ▶ Further, we have

$$MX_i = \begin{pmatrix} | & & | & | & | & \\ Me_1 & \cdots & Me_{i-1} & Mx & Me_{i+1} & \cdots & Me_n \\ | & & | & | & | & \end{pmatrix} = M_i$$

- ▶ Hence,

$$x_i = \det(X_i) = \frac{\det(M_i)}{\det(M)}$$

Proof:

- ▶ Define

$$X_i = \begin{pmatrix} | & & | & | & | & \\ e_1 & \cdots & e_{i-1} & x & e_{i+1} & \cdots & e_n \\ | & & | & | & | & \end{pmatrix}$$

Note that expanding along the i -th column gives that $\det(X_i) = x_i$.

- ▶ Further, we have

$$MX_i = \begin{pmatrix} | & & | & | & | & \\ Me_1 & \cdots & Me_{i-1} & Mx & Me_{i+1} & \cdots & Me_n \\ | & & | & | & | & \end{pmatrix} = M_i$$

- ▶ Hence,

$$x_i = \det(X_i) = \frac{\det(M_i)}{\det(M)}$$

Bounding the Determinant

Let Z be the maximum absolute entry occurring in $\bar{A}$, $\bar{b}$ or c . Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j -th column with vector $\bar{b}$ (for some j).

Observe that

$$|\det(C)|$$

Bounding the Determinant

Let Z be the maximum absolute entry occurring in $\bar{A}$, $\bar{b}$ or c . Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j -th column with vector $\bar{b}$ (for some j).

Observe that

$$|\det(C)| = \left| \sum_{\pi \in S_m} \operatorname{sgn}(\pi) \prod_{1 \leq i \leq m} C_{i\pi(i)} \right|$$

Bounding the Determinant

Let Z be the maximum absolute entry occurring in $\bar{A}, \bar{b}$ or c . Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j -th column with vector $\bar{b}$ (for some j).

Observe that

$$\begin{aligned} |\det(C)| &= \left| \sum_{\pi \in S_m} \operatorname{sgn}(\pi) \prod_{1 \leq i \leq m} C_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_m} \prod_{1 \leq i \leq m} |C_{i\pi(i)}| \end{aligned}$$

Bounding the Determinant

Let Z be the maximum absolute entry occurring in $\bar{A}, \bar{b}$ or c . Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j -th column with vector $\bar{b}$ (for some j).

Observe that

$$\begin{aligned} |\det(C)| &= \left| \sum_{\pi \in S_m} \operatorname{sgn}(\pi) \prod_{1 \leq i \leq m} C_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_m} \prod_{1 \leq i \leq m} |C_{i\pi(i)}| \\ &\leq m! \cdot Z^m . \end{aligned}$$

Bounding the Determinant

Alternatively, Hadamards inequality gives

$$|\det(C)|$$

Bounding the Determinant

Alternatively, Hadamard's inequality gives

$$|\det(C)| \leq \prod_{i=1}^m \|C_{*i}\|$$

Bounding the Determinant

Alternatively, Hadamards inequality gives

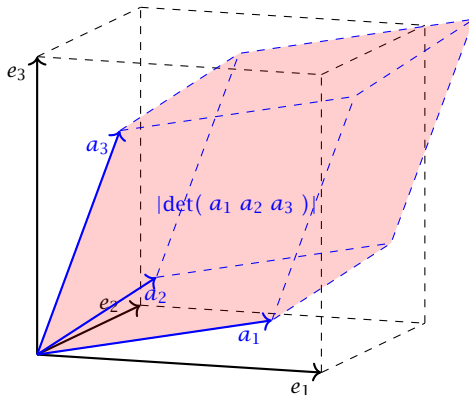
$$|\det(C)| \leq \prod_{i=1}^m \|C_{*i}\| \leq \prod_{i=1}^m (\sqrt{m}Z)$$

Bounding the Determinant

Alternatively, Hadamard's inequality gives

$$\begin{aligned} |\det(C)| &\leq \prod_{i=1}^m \|C_{*i}\| \leq \prod_{i=1}^m (\sqrt{m}Z) \\ &\leq m^{m/2} Z^m . \end{aligned}$$

Hadamards Inequality



Hadamard's inequality says that the volume of the red parallelepiped (**Spat**) is smaller than the volume in the black cube (if $\|e_1\| = \|a_1\|$, $\|e_2\| = \|a_2\|$, $\|e_3\| = \|a_3\|$).

Ensuring Conditions

Given a **standard minimization LP**

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0 \end{array}$$

how can we obtain an LP of the required form?

- ▶ **Compute a lower bound on $c^T x$ for any basic feasible solution.** Add the constraint $c^T x \geq -dZ(m! \cdot Z^m) - 1$. **Note that this constraint is superfluous unless the LP is unbounded.**

Ensuring Conditions

Compute an optimum basis for the new LP.

- ▶ If the cost is $c^T x = -(dZ)(m! \cdot Z^m) - 1$ we know that the original LP is unbounded.
- ▶ Otw. we have an optimum basis.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^T x \geq -dZ(m! \cdot Z^m) - 1$.

We give a routine $\text{SeidelLP}(\mathcal{H}, d)$ that is given a set $\mathcal{H}$ of **explicit, non-degenerate** constraints over d variables, and minimizes $c^T x$ over all feasible points.

In addition it obeys the implicit constraint $c^T x \geq -(dZ)(m! \cdot Z^m) - 1$.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^T x \geq -dZ(m! \cdot Z^m) - 1$.

We give a routine $\text{SeidelLP}(\mathcal{H}, d)$ that is given a set $\mathcal{H}$ of **explicit, non-degenerate** constraints over d variables, and minimizes $c^T x$ over all feasible points.

In addition it obeys the implicit constraint $c^T x \geq -(dZ)(m! \cdot Z^m) - 1$.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^T x \geq -dZ(m! \cdot Z^m) - 1$.

We give a routine $\text{SeidelLP}(\mathcal{H}, d)$ that is given a set $\mathcal{H}$ of **explicit, non-degenerate** constraints over d variables, and minimizes $c^T x$ over all feasible points.

In addition it obeys the implicit constraint $c^T x \geq -(dZ)(m! \cdot Z^m) - 1$.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^T x \geq -dZ(m! \cdot Z^m) - 1$.

We give a routine $\text{SeidelLP}(\mathcal{H}, d)$ that is given a set $\mathcal{H}$ of **explicit, non-degenerate** constraints over d variables, and minimizes $c^T x$ over all feasible points.

In addition it obeys the implicit constraint $c^T x \geq -(dZ)(m! \cdot Z^m) - 1$.

Algorithm 1 SeidelLP($\mathcal{H}, d$)

1: **if** $d = 1$ **then** solve 1-dimensional problem and return;

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$
- 6: **if** $\hat{x}^* = \text{infeasible}$ **then return** infeasible

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$
- 6: **if** $\hat{x}^* = \text{infeasible}$ **then return** infeasible
- 7: **if** $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$
- 6: **if** $\hat{x}^* = \text{infeasible}$ **then return** infeasible
- 7: **if** $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
- 8: // **optimal solution fulfills h with equality, i.e., $a_h^T x = b_h$**

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$
- 6: **if** $\hat{x}^* = \text{infeasible}$ **then return** infeasible
- 7: **if** $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
- 8: // **optimal solution fulfills h with equality, i.e., $a_h^T x = b_h$**
- 9: solve $a_h^T x = b_h$ for some variable x_ℓ ;
- 10: eliminate x_ℓ in constraints from $\hat{\mathcal{H}}$ and in implicit constr.;

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$
- 6: **if** $\hat{x}^* = \text{infeasible}$ **then return** infeasible
- 7: **if** $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
- 8: // **optimal solution fulfills h with equality, i.e., $a_h^T x = b_h$**
- 9: solve $a_h^T x = b_h$ for some variable x_ℓ ;
- 10: eliminate x_ℓ in constraints from $\hat{\mathcal{H}}$ and in implicit constr.;
- 11: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d - 1)$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

- 1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
- 2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
- 3: choose **random** constraint $h \in \mathcal{H}$
- 4: $\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
- 5: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d)$
- 6: **if** $\hat{x}^* = \text{infeasible}$ **then return** infeasible
- 7: **if** $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
- 8: // **optimal solution fulfills h with equality, i.e., $a_h^T x = b_h$**
- 9: solve $a_h^T x = b_h$ for some variable x_ℓ ;
- 10: eliminate x_ℓ in constraints from $\hat{\mathcal{H}}$ and in implicit constr.;
- 11: $\hat{x}^* \leftarrow \text{SeidelLP}(\hat{\mathcal{H}}, d - 1)$
- 12: **if** $\hat{x}^* = \text{infeasible}$ **then**
- 13: **return** infeasible
- 14: **else**
- 15: add the value of x_ℓ to $\hat{x}^*$ and return the solution

8 Seidels LP-algorithm

- ▶ If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(\max\{m, 1\})$.
- ▶ If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d -dimensional vector x .
- ▶ The first recursive call takes time $T(m-1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h .
- ▶ If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m+1)) = \mathcal{O}(dm)$ to eliminate x_ℓ . Then we make a recursive call that takes time $T(m-1, d-1)$.
- ▶ The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- ▶ If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(\max\{m, 1\})$.
- ▶ If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d -dimensional vector x .
- ▶ The first recursive call takes time $T(m - 1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h .
- ▶ If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m + 1)) = \mathcal{O}(dm)$ to eliminate x_ℓ . Then we make a recursive call that takes time $T(m - 1, d - 1)$.
- ▶ The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- ▶ If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(\max\{m, 1\})$.
- ▶ If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d -dimensional vector x .
- ▶ The first recursive call takes time $T(m - 1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h .
- ▶ If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m + 1)) = \mathcal{O}(dm)$ to eliminate x_ℓ . Then we make a recursive call that takes time $T(m - 1, d - 1)$.
- ▶ The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- ▶ If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(\max\{m, 1\})$.
- ▶ If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d -dimensional vector x .
- ▶ The first recursive call takes time $T(m - 1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h .
- ▶ If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m + 1)) = \mathcal{O}(dm)$ to eliminate x_ℓ . Then we make a recursive call that takes time $T(m - 1, d - 1)$.
- ▶ The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- ▶ If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(\max\{m, 1\})$.
- ▶ If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d -dimensional vector x .
- ▶ The first recursive call takes time $T(m - 1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h .
- ▶ If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m + 1)) = \mathcal{O}(dm)$ to eliminate x_ℓ . Then we make a recursive call that takes time $T(m - 1, d - 1)$.
- ▶ The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

This gives the recurrence

$$T(m, d) = \begin{cases} \mathcal{O}(\max\{1, m\}) & \text{if } d = 1 \\ \mathcal{O}(d) & \text{if } d > 1 \text{ and } m = 0 \\ \mathcal{O}(d) + T(m-1, d) + \frac{d}{m}(\mathcal{O}(dm) + T(m-1, d-1)) & \text{otw.} \end{cases}$$

Note that $T(m, d)$ denotes the **expected running time**.

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

$$T(m, d) = \begin{cases} C \max\{1, m\} & \text{if } d = 1 \\ Cd & \text{if } d > 1 \text{ and } m = 0 \\ Cd + T(m - 1, d) + \frac{d}{m}(Cdm + T(m - 1, d - 1)) & \text{otw.} \end{cases}$$

Note that $T(m, d)$ denotes the **expected running time**.

8 Seidels LP-algorithm

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1)$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq C f(1) \max\{1, m\}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq C f(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq C f(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d)$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq C f(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq C f(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq Cf(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq Cf(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d) \max\{1, m\}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq Cf(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq Cf(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d) \max\{1, m\} \text{ for } f(d) \geq d$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq Cf(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq Cf(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d) \max\{1, m\} \text{ for } f(d) \geq d$$

$d > 1; m = 1$:

$$T(1, d) = \mathcal{O}(d) + T(0, d) + d(\mathcal{O}(d) + T(0, d-1))$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq Cf(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq Cf(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d) \max\{1, m\} \text{ for } f(d) \geq d$$

$d > 1; m = 1$:

$$\begin{aligned} T(1, d) &= \mathcal{O}(d) + T(0, d) + d(\mathcal{O}(d) + T(0, d-1)) \\ &\leq Cd + Cd + Cd^2 + dCf(d-1) \end{aligned}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq Cf(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq Cf(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d) \max\{1, m\} \text{ for } f(d) \geq d$$

$d > 1; m = 1$:

$$\begin{aligned} T(1, d) &= \mathcal{O}(d) + T(0, d) + d(\mathcal{O}(d) + T(0, d-1)) \\ &\leq Cd + Cd + Cd^2 + dCf(d-1) \\ &\leq Cf(d) \max\{1, m\} \end{aligned}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$ -notations.

We show $T(m, d) \leq Cf(d) \max\{1, m\}$.

$d = 1$:

$$T(m, 1) \leq C \max\{1, m\} \leq Cf(1) \max\{1, m\} \text{ for } f(1) \geq 1$$

$d > 1; m = 0$:

$$T(0, d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d) \max\{1, m\} \text{ for } f(d) \geq d$$

$d > 1; m = 1$:

$$\begin{aligned} T(1, d) &= \mathcal{O}(d) + T(0, d) + d(\mathcal{O}(d) + T(0, d-1)) \\ &\leq Cd + Cd + Cd^2 + dCf(d-1) \\ &\leq Cf(d) \max\{1, m\} \text{ for } f(d) \geq 3d^2 + df(d-1) \end{aligned}$$

8 Seidels LP-algorithm

$d > 1; m > 1 :$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;

and for $d' = d, m' < m$)

8 Seidels LP-algorithm

$d > 1; m > 1 :$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;

and for $d' = d, m' < m$)

$$T(m, d) = \mathcal{O}(d) + T(m - 1, d) + \frac{d}{m} \left(\mathcal{O}(dm) + T(m - 1, d - 1) \right)$$

8 Seidels LP-algorithm

$d > 1; m > 1 :$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;

and for $d' = d, m' < m$)

$$\begin{aligned} T(m, d) &= \mathcal{O}(d) + T(m-1, d) + \frac{d}{m} \left(\mathcal{O}(dm) + T(m-1, d-1) \right) \\ &\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m} Cf(d-1)(m-1) \end{aligned}$$

8 Seidels LP-algorithm

$d > 1; m > 1 :$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;

and for $d' = d, m' < m$)

$$\begin{aligned} T(m, d) &= \mathcal{O}(d) + T(m-1, d) + \frac{d}{m} \left(\mathcal{O}(dm) + T(m-1, d-1) \right) \\ &\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m} Cf(d-1)(m-1) \\ &\leq 2Cd^2 + Cf(d)(m-1) + dCf(d-1) \end{aligned}$$

8 Seidels LP-algorithm

$d > 1; m > 1 :$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;

and for $d' = d, m' < m$)

$$\begin{aligned} T(m, d) &= \mathcal{O}(d) + T(m-1, d) + \frac{d}{m} \left(\mathcal{O}(dm) + T(m-1, d-1) \right) \\ &\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m} Cf(d-1)(m-1) \\ &\leq 2Cd^2 + Cf(d)(m-1) + dCf(d-1) \\ &\leq Cf(d)m \end{aligned}$$

8 Seidels LP-algorithm

$d > 1; m > 1 :$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;

and for $d' = d, m' < m$)

$$\begin{aligned} T(m, d) &= \mathcal{O}(d) + T(m-1, d) + \frac{d}{m} \left(\mathcal{O}(dm) + T(m-1, d-1) \right) \\ &\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m} Cf(d-1)(m-1) \\ &\leq 2Cd^2 + Cf(d)(m-1) + dCf(d-1) \\ &\leq Cf(d)m \end{aligned}$$

if $f(d) \geq df(d-1) + 2d^2$.

8 Seidels LP-algorithm

- ▶ Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

8 Seidels LP-algorithm

- ▶ Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$f(d)$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$f(d) = 3d^2 + df(d-1)$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned} f(d) &= 3d^2 + df(d-1) \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1)f(d-2) \right] \end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned}f(d) &= 3d^2 + df(d-1) \\&= 3d^2 + d \left[3(d-1)^2 + (d-1)f(d-2) \right] \\&= 3d^2 + d \left[3(d-1)^2 + (d-1) \left[3(d-2)^2 + (d-2)f(d-3) \right] \right]\end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned} f(d) &= 3d^2 + df(d-1) \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1)f(d-2) \right] \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1) \left[3(d-2)^2 + (d-2)f(d-3) \right] \right] \\ &= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\ &\quad + 3d(d-1)(d-2) \cdot \dots \cdot 4 \cdot 3 \cdot 2 \cdot 1^2 \end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned} f(d) &= 3d^2 + df(d-1) \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1)f(d-2) \right] \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1) \left[3(d-2)^2 + (d-2)f(d-3) \right] \right] \\ &= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\ &\quad + 3d(d-1)(d-2) \cdot \dots \cdot 4 \cdot 3 \cdot 2 \cdot 1^2 \\ &= 3d! \left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots \right) \end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned} f(d) &= 3d^2 + df(d-1) \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1)f(d-2) \right] \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1) \left[3(d-2)^2 + (d-2)f(d-3) \right] \right] \\ &= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\ &\quad + 3d(d-1)(d-2) \cdot \dots \cdot 4 \cdot 3 \cdot 2 \cdot 1^2 \\ &= 3d! \left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots \right) \\ &= \mathcal{O}(d!) \end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned} f(d) &= 3d^2 + df(d-1) \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1)f(d-2) \right] \\ &= 3d^2 + d \left[3(d-1)^2 + (d-1) \left[3(d-2)^2 + (d-2)f(d-3) \right] \right] \\ &= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\ &\quad + 3d(d-1)(d-2) \cdot \dots \cdot 4 \cdot 3 \cdot 2 \cdot 1^2 \\ &= 3d! \left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots \right) \\ &= \mathcal{O}(d!) \end{aligned}$$

since $\sum_{i \geq 1} \frac{i^2}{i!}$ is a constant.

Complexity

LP Feasibility Problem (LP feasibility A)

Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$. Does there exist $x \in \mathbb{R}^n$ with $Ax \leq b$, $x \geq 0$?

LP Feasibility Problem (LP feasibility B)

Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$. Find $x \in \mathbb{R}^n$ with $Ax \leq b$, $x \geq 0$!

LP Optimization A

Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$. What is the maximum value of $c^T x$ for a feasible point $x \in \mathbb{R}^n$?

LP Optimization B

Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$. Return feasible point $x \in \mathbb{R}^n$ with maximum value of $c^T x$?

The Bit Model

Input size

- ▶ The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- ▶ Let for an $m \times n$ matrix M , $L(M)$ denote the number of bits required to encode all the numbers in M .

$$\langle M \rangle := \sum_{i,j} \lceil \log_2(|m_{ij}|) \rceil + 1$$

- ▶ In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- ▶ Then the input length is $L = \Theta(\langle A \rangle + \langle b \rangle)$.

The Bit Model

Input size

- ▶ The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- ▶ Let for an $m \times n$ matrix M , $L(M)$ denote the number of bits required to encode all the numbers in M .

$$\langle M \rangle := \sum_{i,j} \lceil \log_2(|m_{ij}|) \rceil + 1$$

- ▶ In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- ▶ Then the input length is $L = \Theta(\langle A \rangle + \langle b \rangle)$.

The Bit Model

Input size

- ▶ The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- ▶ Let for an $m \times n$ matrix M , $L(M)$ denote the number of bits required to encode all the numbers in M .

$$\langle M \rangle := \sum_{i,j} \lceil \log_2(|m_{ij}|) \rceil + 1$$

- ▶ In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- ▶ Then the input length is $L = \Theta(\langle A \rangle + \langle b \rangle)$.

The Bit Model

Input size

- ▶ The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- ▶ Let for an $m \times n$ matrix M , $L(M)$ denote the number of bits required to encode all the numbers in M .

$$\langle M \rangle := \sum_{i,j} \lceil \log_2(|m_{ij}|) \rceil + 1$$

- ▶ In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- ▶ Then the input length is $L = \Theta(\langle A \rangle + \langle b \rangle)$.

- ▶ In the following we sometimes refer to $L := \langle A \rangle + \langle b \rangle$ as the input size (even though the real input size is something in $\Theta(\langle A \rangle + \langle b \rangle)$).
- ▶ Sometimes we may also refer to $L := \langle A \rangle + \langle b \rangle + n \log_2 n$ as the input size. Note that $n \log_2 n = \Theta(\langle A \rangle + \langle b \rangle)$.
- ▶ In order to show that LP-decision is in NP we show that if there is a solution x then there exists a small solution for which feasibility can be verified in polynomial time (polynomial in L).

Suppose that $\bar{A}x = b; x \geq 0$ is feasible.

Then there exists a basic feasible solution. This means a set B of basic variables such that

$$x_B = \bar{A}_B^{-1}b$$

and all other entries in x are 0.

Suppose that $\bar{A}x = b$; $x \geq 0$ is feasible.

Then there exists a basic feasible solution. This means a set B of basic variables such that

$$x_B = \bar{A}_B^{-1}b$$

and all other entries in x are 0.

Size of a Basic Feasible Solution

- ▶ A : original input matrix
- ▶ $\bar{A}$: transformation of A into standard form
- ▶ $\bar{A}_B$: submatrix of $\bar{A}$ corresponding to basis B

Lemma 47

Let $\bar{A}_B \in \mathbb{Z}^{m \times m}$ and $b \in \mathbb{Z}^m$. Define $L = \langle A \rangle + \langle b \rangle + n \log_2 n$. Then a solution to $\bar{A}_B x_B = b$ has rational components x_j of the form $\frac{D_j}{D}$, where $|D_j| \leq 2^L$ and $|D| \leq 2^L$.

Proof:

Cramer's rule says that we can compute x_j as

$$x_j = \frac{\det(\bar{A}_B^j)}{\det(\bar{A}_B)}$$

where $\bar{A}_B^j$ is the matrix obtained from $\bar{A}_B$ by replacing the j -th column by the vector b .

Size of a Basic Feasible Solution

- ▶ A : original input matrix
- ▶ $\bar{A}$: transformation of A into standard form
- ▶ $\bar{A}_B$: submatrix of $\bar{A}$ corresponding to basis B

Lemma 47

Let $\bar{A}_B \in \mathbb{Z}^{m \times m}$ and $b \in \mathbb{Z}^m$. Define $L = \langle A \rangle + \langle b \rangle + n \log_2 n$. Then a solution to $\bar{A}_B x_B = b$ has rational components x_j of the form $\frac{D_j}{D}$, where $|D_j| \leq 2^L$ and $|D| \leq 2^L$.

Proof:

Cramers rules says that we can compute x_j as

$$x_j = \frac{\det(\bar{A}_B^j)}{\det(\bar{A}_B)}$$

where $\bar{A}_B^j$ is the matrix obtained from $\bar{A}_B$ by replacing the j -th column by the vector b .

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$|\det(X)|$$

Bounding the Determinant

Let $X = \tilde{A}_B$. Then

$$|\det(X)| = |\det(\tilde{X})|$$

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$\begin{aligned} |\det(X)| &= |\det(\tilde{X})| \\ &= \left| \sum_{\pi \in S_{\tilde{n}}} \text{sgn}(\pi) \prod_{1 \leq i \leq \tilde{n}} \tilde{X}_{i\pi(i)} \right| \end{aligned}$$

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$\begin{aligned} |\det(X)| &= |\det(\tilde{X})| \\ &= \left| \sum_{\pi \in S_{\tilde{n}}} \operatorname{sgn}(\pi) \prod_{1 \leq i \leq \tilde{n}} \tilde{X}_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_{\tilde{n}}} \prod_{1 \leq i \leq \tilde{n}} |\tilde{X}_{i\pi(i)}| \end{aligned}$$

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$\begin{aligned} |\det(X)| &= |\det(\tilde{X})| \\ &= \left| \sum_{\pi \in S_{\tilde{n}}} \text{sgn}(\pi) \prod_{1 \leq i \leq \tilde{n}} \tilde{X}_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_{\tilde{n}}} \prod_{1 \leq i \leq \tilde{n}} |\tilde{X}_{i\pi(i)}| \\ &\leq n! \cdot 2^{\langle A \rangle + \langle b \rangle} \end{aligned}$$

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$\begin{aligned} |\det(X)| &= |\det(\tilde{X})| \\ &= \left| \sum_{\pi \in S_{\tilde{n}}} \text{sgn}(\pi) \prod_{1 \leq i \leq \tilde{n}} \tilde{X}_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_{\tilde{n}}} \prod_{1 \leq i \leq \tilde{n}} |\tilde{X}_{i\pi(i)}| \\ &\leq n! \cdot 2^{\langle A \rangle + \langle b \rangle} \leq 2^L . \end{aligned}$$

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$\begin{aligned} |\det(X)| &= |\det(\tilde{X})| \\ &= \left| \sum_{\pi \in S_{\tilde{n}}} \text{sgn}(\pi) \prod_{1 \leq i \leq \tilde{n}} \tilde{X}_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_{\tilde{n}}} \prod_{1 \leq i \leq \tilde{n}} |\tilde{X}_{i\pi(i)}| \\ &\leq n! \cdot 2^{\langle A \rangle + \langle b \rangle} \leq 2^L. \end{aligned}$$

Here $\tilde{X}$ is an $\tilde{n} \times \tilde{n}$ submatrix of A with $\tilde{n} \leq n$.

Bounding the Determinant

Let $X = \bar{A}_B$. Then

$$\begin{aligned} |\det(X)| &= |\det(\bar{X})| \\ &= \left| \sum_{\pi \in S_{\tilde{n}}} \text{sgn}(\pi) \prod_{1 \leq i \leq \tilde{n}} \bar{X}_{i\pi(i)} \right| \\ &\leq \sum_{\pi \in S_{\tilde{n}}} \prod_{1 \leq i \leq \tilde{n}} |\bar{X}_{i\pi(i)}| \\ &\leq n! \cdot 2^{\langle A \rangle + \langle b \rangle} \leq 2^L. \end{aligned}$$

Here $\bar{X}$ is an $\tilde{n} \times \tilde{n}$ submatrix of A with $\tilde{n} \leq n$.

Analogously for $\det(A_B^j)$.

Reducing LP-solving to LP decision.

Given an LP $\max\{c^T x \mid Ax \leq b; x \geq 0\}$ do a **binary search** for the optimum solution

(Add constraint $c^T x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2 \left(\frac{2n2^{2L'}}{1/2^{L'}} \right) = \mathcal{O}(L') ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = \langle A \rangle + \langle b \rangle + \langle c \rangle + n \log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^T x \mid Ax \leq b; x \geq 0\}$ do a **binary search** for the optimum solution

(Add constraint $c^T x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2 \left(\frac{2n2^{2L'}}{1/2^{L'}} \right) = \mathcal{O}(L') ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = \langle A \rangle + \langle b \rangle + \langle c \rangle + n \log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^T x \mid Ax \leq b; x \geq 0\}$ do a **binary search** for the optimum solution

(Add constraint $c^T x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2 \left(\frac{2n2^{2L'}}{1/2^{L'}} \right) = \mathcal{O}(L') ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = \langle A \rangle + \langle b \rangle + \langle c \rangle + n \log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^T x \mid Ax \leq b; x \geq 0\}$ do a **binary search** for the optimum solution

(Add constraint $c^T x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2 \left(\frac{2n2^{2L'}}{1/2^{L'}} \right) = \mathcal{O}(L') ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = \langle A \rangle + \langle b \rangle + \langle c \rangle + n \log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^T x \mid Ax \leq b; x \geq 0\}$ do a **binary search** for the optimum solution

(Add constraint $c^T x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2 \left(\frac{2n2^{2L'}}{1/2^{L'}} \right) = \mathcal{O}(L') ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = \langle A \rangle + \langle b \rangle + \langle c \rangle + n \log_2 n$ (it also includes the encoding size of c).

How do we detect whether the LP is unbounded?

Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a basic feasible solution.

We can add a constraint $c^T x \geq M_{\max} + 1$ and check for feasibility.

How do we detect whether the LP is unbounded?

Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a **basic feasible solution**.

We can add a constraint $c^T x \geq M_{\max} + 1$ and check for feasibility.

How do we detect whether the LP is unbounded?

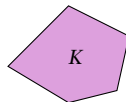
Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a **basic feasible solution**.

We can add a constraint $c^T x \geq M_{\max} + 1$ and check for feasibility.

Ellipsoid Method

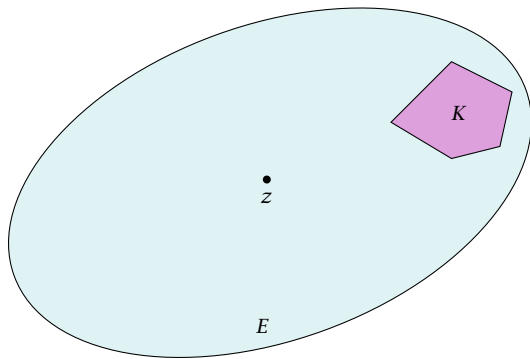
Ellipsoid Method

- ▶ Let K be a convex set.



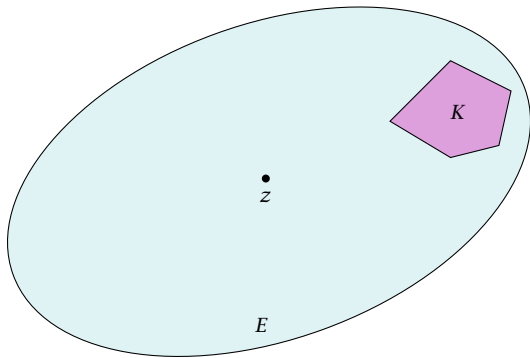
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.



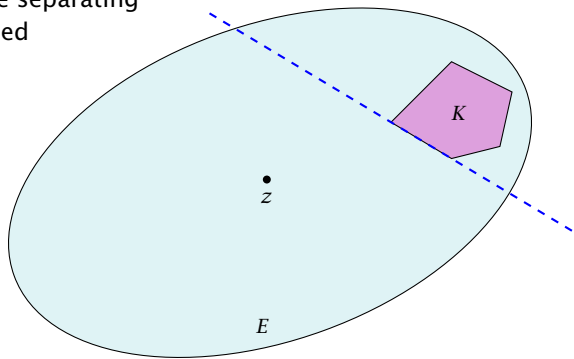
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.



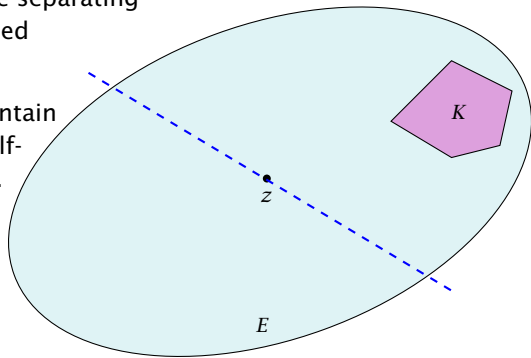
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.
- ▶ Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).



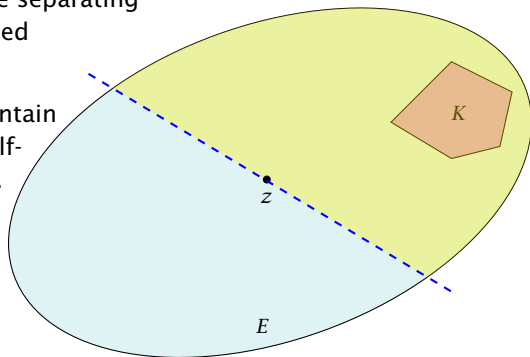
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.
- ▶ Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- ▶ Shift hyperplane to contain node z . H denotes half-space that contains K .



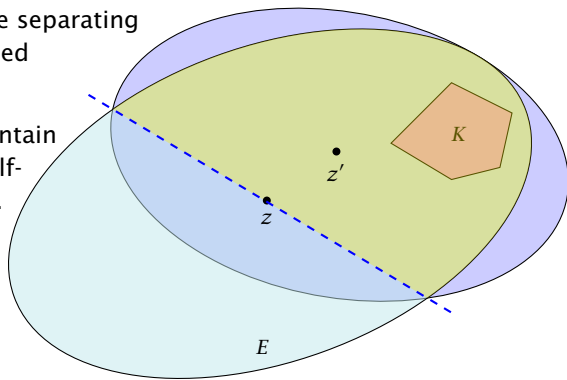
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.
- ▶ Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- ▶ Shift hyperplane to contain node z . H denotes half-space that contains K .



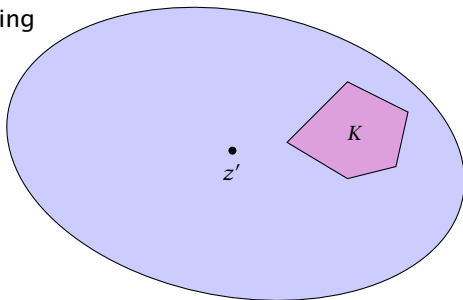
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.
- ▶ Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- ▶ Shift hyperplane to contain node z . H denotes half-space that contains K .
- ▶ Compute (smallest) ellipsoid E' that contains $E \cap H$.



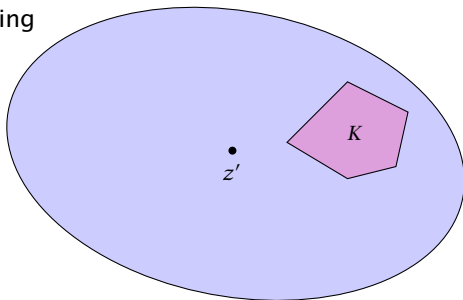
Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.
- ▶ Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- ▶ Shift hyperplane to contain node z . H denotes half-space that contains K .
- ▶ Compute (smallest) ellipsoid E' that contains $E \cap H$.



Ellipsoid Method

- ▶ Let K be a convex set.
- ▶ Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- ▶ If center $z \in K$ STOP.
- ▶ Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- ▶ Shift hyperplane to contain node z . H denotes half-space that contains K .
- ▶ Compute (smallest) ellipsoid E' that contains $E \cap H$.
- ▶ REPEAT



Issues/Questions:

- ▶ How do you choose the first Ellipsoid? What is its volume?
- ▶ How do you measure progress? By how much does the volume decrease in each iteration?
- ▶ When can you stop? What is the minimum volume of a non-empty polytop?

Definition 48

A mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ with $f(x) = Lx + t$, where L is an invertible matrix is called an **affine transformation**.

Definition 49

A ball in $\mathbb{R}^n$ with center c and radius r is given by

$$\begin{aligned} B(c, r) &= \{x \mid (x - c)^T (x - c) \leq r^2\} \\ &= \{x \mid \sum_i (x - c)_i^2 / r^2 \leq 1\} \end{aligned}$$

$B(0, 1)$ is called the **unit ball**.

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$f(B(0, 1))$$

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$f(B(0, 1)) = \{f(x) \mid x \in B(0, 1)\}$$

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0, 1)) &= \{f(x) \mid x \in B(0, 1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y - t) \in B(0, 1)\} \end{aligned}$$

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y - t) \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid (y - t)^T L^{-1^T} L^{-1} (y - t) \leq 1\} \end{aligned}$$

Definition 50

An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y - t) \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid (y - t)^T L^{-1^T} L^{-1} (y - t) \leq 1\} \\ &= \{y \in \mathbb{R}^n \mid (y - t)^T Q^{-1} (y - t) \leq 1\} \end{aligned}$$

Definition 50

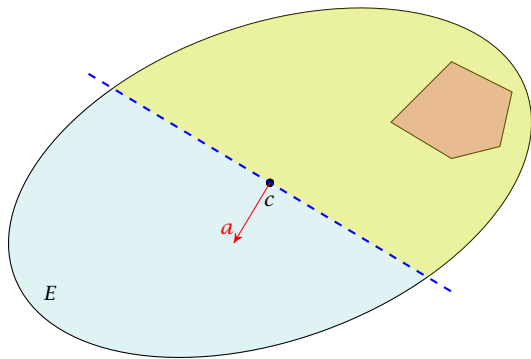
An affine transformation of the unit ball is called an **ellipsoid**.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y - t) \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid (y - t)^T L^{-1^T} L^{-1} (y - t) \leq 1\} \\ &= \{y \in \mathbb{R}^n \mid (y - t)^T Q^{-1} (y - t) \leq 1\} \end{aligned}$$

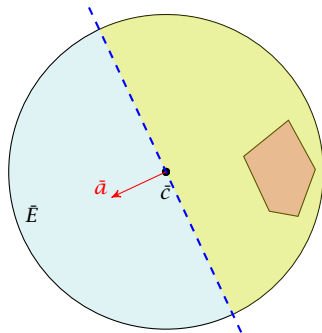
where $Q = LL^T$ is an invertible matrix.

How to Compute the New Ellipsoid



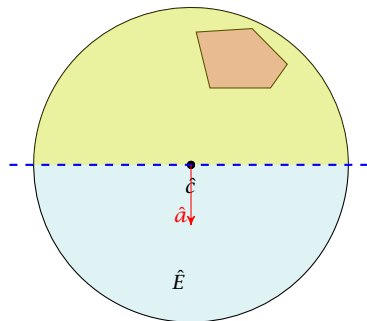
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.



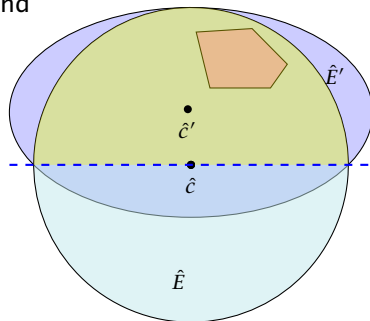
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .



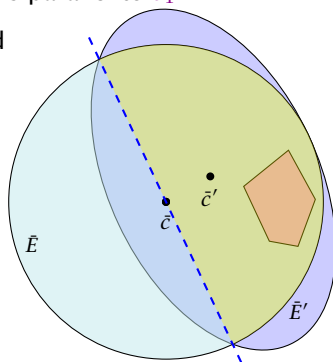
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.



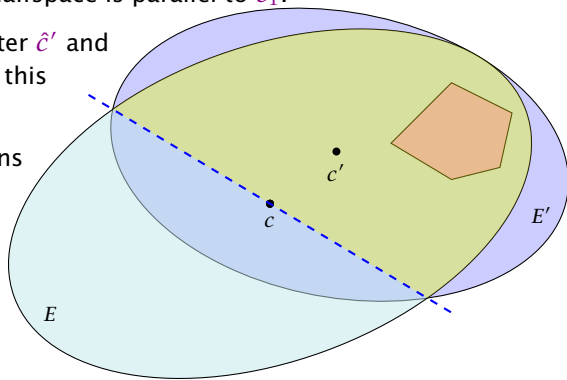
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\tilde{c}'$ and the new matrix $\tilde{Q}'$ for this simplified setting.
- ▶ Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E .

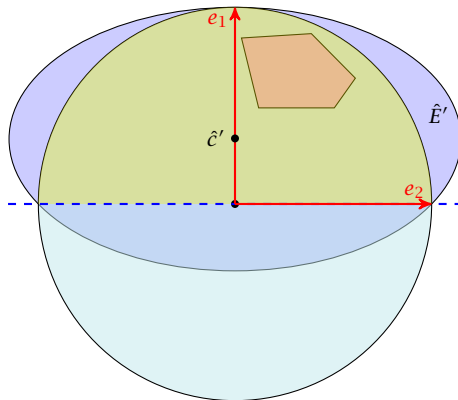


How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- ▶ Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E .

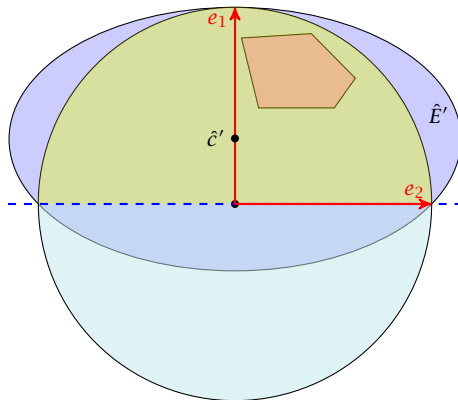


The Easy Case



- ▶ The new center lies on axis x_1 . Hence, $\hat{c}' = te_1$ for $t > 0$.
- ▶ The vectors $e_1, e_2, \dots$ have to fulfill the ellipsoid constraint with equality. Hence $(e_i - \hat{c}')^T \hat{Q}'^{-1} (e_i - \hat{c}') = 1$.

The Easy Case



- ▶ The new center lies on axis x_1 . Hence, $\hat{c}' = te_1$ for $t > 0$.
- ▶ The vectors $e_1, e_2, \dots$ have to fulfill the ellipsoid constraint with equality. Hence $(e_i - \hat{c}')^T \hat{Q}'^{-1} (e_i - \hat{c}') = 1$.

The Easy Case

- ▶ To obtain the matrix $\hat{Q}'^{-1}$ for our ellipsoid $\hat{E}'$ note that $\hat{E}'$ is **axis-parallel**.
- ▶ Let a denote the radius along the x_1 -axis and let b denote the (common) radius for the other axes.
- ▶ The matrix

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

maps the unit ball (via function $\hat{f}'(x) = \hat{L}'x$) to an axis-parallel ellipsoid with radius a in direction x_1 and b in all other directions.

The Easy Case

- ▶ To obtain the matrix $\hat{Q}'^{-1}$ for our ellipsoid $\hat{E}'$ note that $\hat{E}'$ is **axis-parallel**.
- ▶ Let a denote the radius along the x_1 -axis and let b denote the (common) radius for the other axes.
- ▶ The matrix

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

maps the unit ball (via function $\hat{f}'(x) = \hat{L}'x$) to an axis-parallel ellipsoid with radius a in direction x_1 and b in all other directions.

The Easy Case

- ▶ To obtain the matrix $\hat{Q}'^{-1}$ for our ellipsoid $\hat{E}'$ note that $\hat{E}'$ is **axis-parallel**.
- ▶ Let a denote the radius along the x_1 -axis and let b denote the (common) radius for the other axes.
- ▶ The matrix

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

maps the unit ball (via function $\hat{f}'(x) = \hat{L}'x$) to an axis-parallel ellipsoid with radius a in direction x_1 and b in all other directions.

The Easy Case

- As $\hat{Q}' = \hat{L}'\hat{L}'^t$ the matrix $\hat{Q}'^{-1}$ is of the form

$$\hat{Q}'^{-1} = \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix}$$

The Easy Case

- ▶ $(e_1 - \hat{c}')^T \hat{Q}'^{-1} (e_1 - \hat{c}') = 1$ gives

$$\begin{pmatrix} 1-t \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} 1-t \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- ▶ This gives $(1-t)^2 = a^2$.

The Easy Case

- For $i \neq 1$ the equation $(e_i - \hat{c}')^T \hat{Q}'^{-1} (e_i - \hat{c}') = 1$ looks like (here $i = 2$)

$$\begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- This gives $\frac{t^2}{a^2} + \frac{1}{b^2} = 1$, and hence

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2}$$

The Easy Case

- For $i \neq 1$ the equation $(e_i - \hat{c}')^T \hat{Q}'^{-1} (e_i - \hat{c}') = 1$ looks like (here $i = 2$)

$$\begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- This gives $\frac{t^2}{a^2} + \frac{1}{b^2} = 1$, and hence

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2} = 1 - \frac{t^2}{(1-t)^2}$$

The Easy Case

- ▶ For $i \neq 1$ the equation $(e_i - \hat{c}')^T \hat{Q}'^{-1} (e_i - \hat{c}') = 1$ looks like (here $i = 2$)

$$\begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^T \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- ▶ This gives $\frac{t^2}{a^2} + \frac{1}{b^2} = 1$, and hence

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2} = 1 - \frac{t^2}{(1-t)^2} = \frac{1-2t}{(1-t)^2}$$

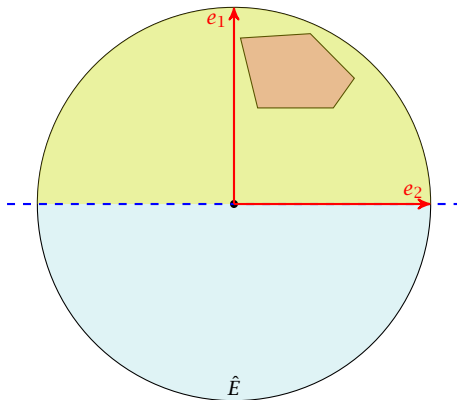
Summary

So far we have

$$a = 1 - t \quad \text{and} \quad b = \frac{1 - t}{\sqrt{1 - 2t}}$$

The Easy Case

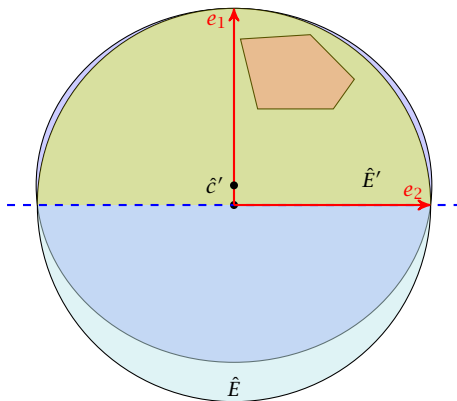
We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

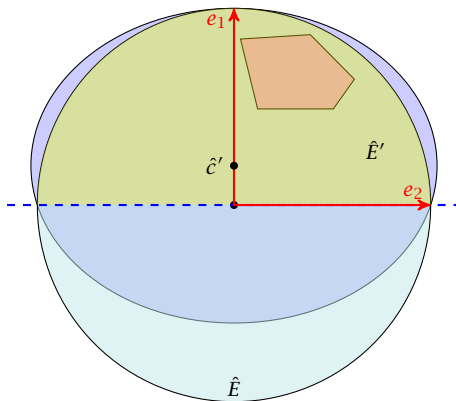
We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

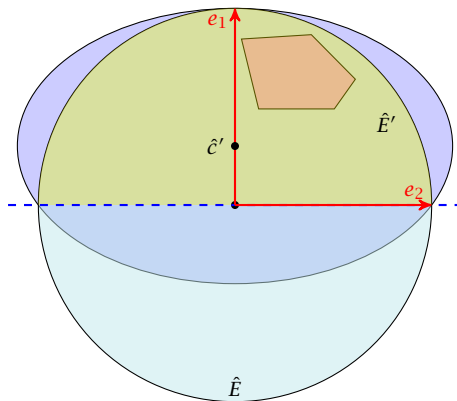
We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

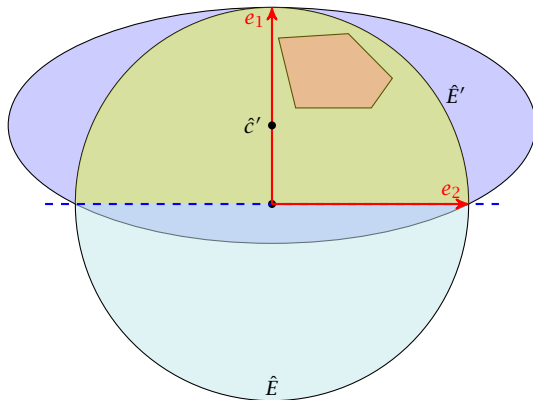
We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

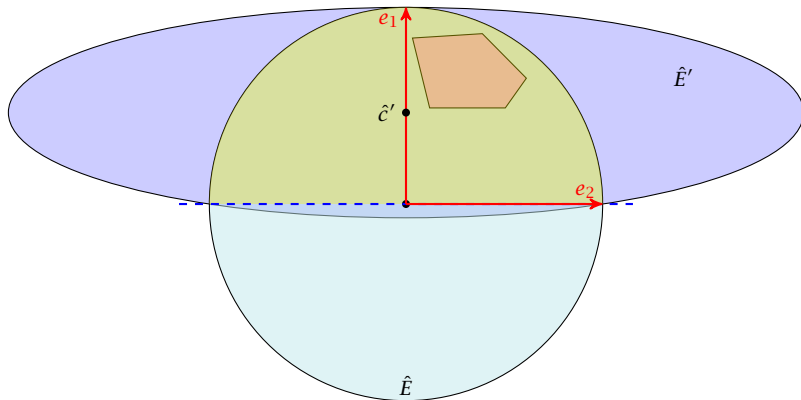
We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

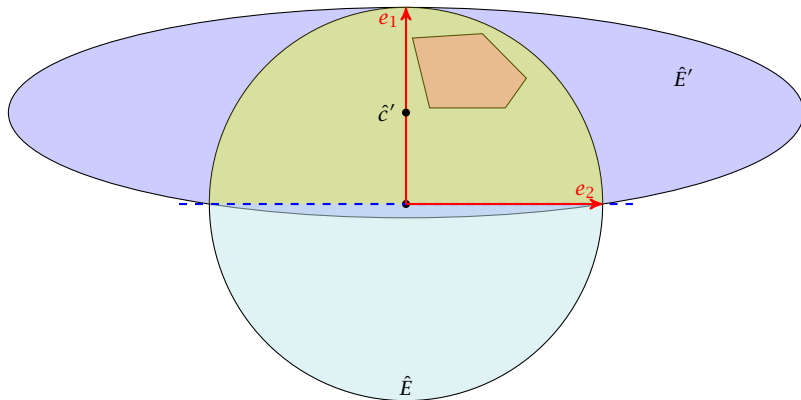
We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

We still have many choices for t :



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

We want to choose t such that the volume of $\hat{E}'$ is minimal.

Lemma 51

Let L be an affine transformation and $K \subseteq \mathbb{R}^n$. Then

$$\text{vol}(L(K)) = |\det(L)| \cdot \text{vol}(K) .$$

The Easy Case

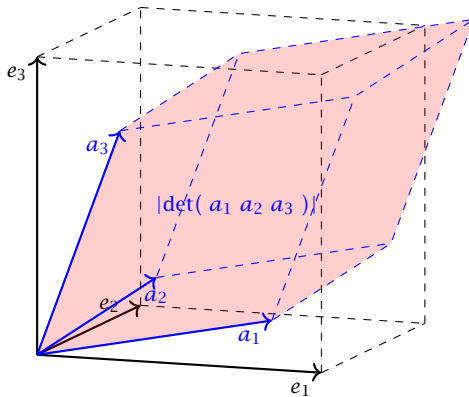
We want to choose t such that the volume of $\hat{E}'$ is minimal.

Lemma 51

Let L be an affine transformation and $K \subseteq \mathbb{R}^n$. Then

$$\text{vol}(L(K)) = |\det(L)| \cdot \text{vol}(K) .$$

n-dimensional volume



The Easy Case

- ▶ We want to choose t such that the volume of $\hat{E}'$ is minimal.

$$\text{vol}(\hat{E}') = \text{vol}(B(0,1)) \cdot |\det(\hat{L}')| ,$$

- ▶ Recall that

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

- ▶ Note that a and b in the above equations depend on t , by the previous equations.

The Easy Case

- ▶ We want to choose t such that the volume of $\hat{E}'$ is minimal.

$$\text{vol}(\hat{E}') = \text{vol}(B(0, 1)) \cdot |\det(\hat{L}')| ,$$

- ▶ Recall that

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

- ▶ Note that a and b in the above equations depend on t , by the previous equations.

The Easy Case

- ▶ We want to choose t such that the volume of $\hat{E}'$ is minimal.

$$\text{vol}(\hat{E}') = \text{vol}(B(0,1)) \cdot |\det(\hat{L}')| ,$$

- ▶ Recall that

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

- ▶ Note that a and b in the above equations depend on t , by the previous equations.

The Easy Case

$$\text{vol}(\hat{E}')$$

The Easy Case

$$\text{vol}(\hat{E}') = \text{vol}(B(0, 1)) \cdot |\det(\hat{L}')|$$

The Easy Case

$$\begin{aligned}\text{vol}(\hat{E}') &= \text{vol}(B(0, 1)) \cdot |\det(\hat{L}')| \\ &= \text{vol}(B(0, 1)) \cdot ab^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\text{vol}(\hat{E}') &= \text{vol}(B(0, 1)) \cdot |\det(\hat{L}')| \\ &= \text{vol}(B(0, 1)) \cdot ab^{n-1} \\ &= \text{vol}(B(0, 1)) \cdot (1 - t) \cdot \left(\frac{1 - t}{\sqrt{1 - 2t}} \right)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\text{vol}(\hat{E}') &= \text{vol}(B(0, 1)) \cdot |\det(\hat{L}')| \\ &= \text{vol}(B(0, 1)) \cdot ab^{n-1} \\ &= \text{vol}(B(0, 1)) \cdot (1-t) \cdot \left(\frac{1-t}{\sqrt{1-2t}} \right)^{n-1} \\ &= \text{vol}(B(0, 1)) \cdot \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\end{aligned}$$

The Easy Case

$$\begin{aligned}\text{vol}(\hat{E}') &= \text{vol}(B(0, 1)) \cdot |\det(\hat{L}')| \\ &= \text{vol}(B(0, 1)) \cdot ab^{n-1} \\ &= \text{vol}(B(0, 1)) \cdot (1 - t) \cdot \left(\frac{1 - t}{\sqrt{1 - 2t}} \right)^{n-1} \\ &= \text{vol}(B(0, 1)) \cdot \frac{(1 - t)^n}{(\sqrt{1 - 2t})^{n-1}}\end{aligned}$$

We use the shortcut $\Phi := \text{vol}(B(0, 1))$.

The Easy Case

$$\frac{d \operatorname{vol}(\hat{E}')}{d t}$$

The Easy Case

$$\frac{d \operatorname{vol}(\hat{E}')}{d t} = \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right)$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\ &= \frac{\Phi}{N^2}\end{aligned}$$

$N = \text{denominator}$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\ &= \frac{\Phi}{N^2} \cdot \left(\underbrace{(-1) \cdot n(1-t)^{n-1}}_{\text{derivative of numerator}} \right)\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\ &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot \underbrace{(\sqrt{1-2t})^{n-1}}_{\text{denominator}} \right)\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\ &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot (\sqrt{1-2t})^{n-1} \right. \\ &\quad \left. - (n-1)(\sqrt{1-2t})^{n-2} \right) \\ &\quad \text{outer derivative}\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\ &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot (\sqrt{1-2t})^{n-1} \right. \\ &\quad \left. - (n-1)(\sqrt{1-2t})^{n-2} \cdot \frac{1}{2\sqrt{1-2t}} \cdot (-2) \right)\end{aligned}$$

inner derivative

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\ &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot (\sqrt{1-2t})^{n-1} \right. \\ &\quad \left. - (n-1)(\sqrt{1-2t})^{n-2} \cdot \frac{1}{2\sqrt{1-2t}} \cdot (-2) \cdot (1-t)^n \right) \\ &\quad \text{numerator}\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\&= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot (\sqrt{1-2t})^{n-1} \right. \\&\quad \left. - (n-1)(\sqrt{1-2t})^{n-2} \cdot \frac{1}{2\sqrt{1-2t}} \cdot (-2) \cdot (1-t)^n \right) \\&= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\&= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot \frac{1-2t}{\sqrt{1-2t}} \cdot (\sqrt{1-2t})^{n-1} \right. \\&\quad \left. - (n-1)(\sqrt{1-2t})^{n-2} \cdot \frac{1}{2\sqrt{1-2t}} \cdot (-2) \cdot (1-t)^n \right) \\&= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\&= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n(1-t)^{n-1} \cdot \frac{1-2t}{(\sqrt{1-2t})^{n-1}} \right. \\&\quad \left. - (n-1) \frac{1}{2\sqrt{1-2t}} \cdot (-2) \cdot (1-t)^n \right) \\&= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\&= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n \cancel{(1-t)^{n-1}} \cdot \cancel{(\sqrt{1-2t})^{n-1}}^{1-2t} \right. \\&\quad \left. - (n-1) \cancel{(\sqrt{1-2t})^{n-2}} \cdot \frac{1}{2\sqrt{1-2t}} \cdot (-2) \cdot (1-t)^n \right) \\&= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\&= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n \cancel{(1-t)^{n-1}} \cdot \cancel{(\sqrt{1-2t})^{n-1}}^{1-2t} \right. \\&\quad \left. - (n-1) \cancel{(\sqrt{1-2t})^{n-2}} \cdot \frac{1}{2\sqrt{1-2t}} \cdot (-2) \cdot \cancel{(1-t)^n}^{1-t} \right) \\&= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}
 \frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\
 &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n \cancel{(1-t)^{n-1}} \cdot \cancel{(\sqrt{1-2t})^{n-1}}^{1-2t} \right. \\
 &\quad \left. \cancel{(n-1)(\sqrt{1-2t})^{n-2}} \cdot \frac{1}{2\sqrt{1-2t}} \cdot \cancel{(-2)} \cdot \cancel{(1-t)^n}^{1-t} \right) \\
 &= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1}
 \end{aligned}$$

The Easy Case

$$\begin{aligned}
 \frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\
 &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n \cancel{(1-t)^{n-1}} \cdot \cancel{(\sqrt{1-2t})^{n-1}}^{1-2t} \right. \\
 &\quad \left. \cancel{(n-1)(\sqrt{1-2t})^{n-2}} \cdot \frac{1}{2\sqrt{1-2t}} \cdot \cancel{(-2)} \cdot \cancel{(1-t)^n}^{1-t} \right) \\
 &= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1} \\
 &\quad \cdot \left((n-1)(1-t) - n(1-2t) \right)
 \end{aligned}$$

The Easy Case

$$\begin{aligned}
 \frac{d \operatorname{vol}(\hat{E}')}{d t} &= \frac{d}{d t} \left(\Phi \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}} \right) \\
 &= \frac{\Phi}{N^2} \cdot \left((-1) \cdot n \cancel{(1-t)^{n-1}} \cdot \cancel{(\sqrt{1-2t})^{n-1}}^{1-2t} \right. \\
 &\quad \left. \cancel{(n-1)(\sqrt{1-2t})^{n-2}} \cdot \frac{1}{2\sqrt{1-2t}} \cdot \cancel{(-2)} \cdot \cancel{(1-t)^n}^{1-t} \right) \\
 &= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1} \\
 &\quad \cdot \left((n-1)(1-t) - n(1-2t) \right) \\
 &= \frac{\Phi}{N^2} \cdot (\sqrt{1-2t})^{n-3} \cdot (1-t)^{n-1} \cdot \left((n+1)t - 1 \right)
 \end{aligned}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

a

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b =$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b , observe that

$$b^2$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b , observe that

$$b^2 = \frac{(1-t)^2}{1-2t}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b , observe that

$$b^2 = \frac{(1-t)^2}{1-2t} = \frac{(1 - \frac{1}{n+1})^2}{1 - \frac{2}{n+1}}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b , observe that

$$b^2 = \frac{(1-t)^2}{1-2t} = \frac{(1-\frac{1}{n+1})^2}{1-\frac{2}{n+1}} = \frac{(\frac{n}{n+1})^2}{\frac{n-1}{n+1}}$$

The Easy Case

- ▶ We obtain the minimum for $t = \frac{1}{n+1}$.
- ▶ For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b , observe that

$$b^2 = \frac{(1-t)^2}{1-2t} = \frac{(1-\frac{1}{n+1})^2}{1-\frac{2}{n+1}} = \frac{(\frac{n}{n+1})^2}{\frac{n-1}{n+1}} = \frac{n^2}{n^2-1}$$

The Easy Case

Let $\gamma_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\gamma_n^2$$

The Easy Case

Let $\gamma_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\gamma_n^2 = \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1}$$

The Easy Case

Let $y_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned} y_n^2 &= \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1} \\ &= \left(1 - \frac{1}{n+1}\right)^2 \left(1 + \frac{1}{(n-1)(n+1)}\right)^{n-1} \end{aligned}$$

The Easy Case

Let $y_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned} y_n^2 &= \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1} \\ &= \left(1 - \frac{1}{n+1}\right)^2 \left(1 + \frac{1}{(n-1)(n+1)}\right)^{n-1} \\ &\leq e^{-2\frac{1}{n+1}} \cdot e^{\frac{1}{n+1}} \end{aligned}$$

The Easy Case

Let $y_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned} y_n^2 &= \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1} \\ &= \left(1 - \frac{1}{n+1}\right)^2 \left(1 + \frac{1}{(n-1)(n+1)}\right)^{n-1} \\ &\leq e^{-2\frac{1}{n+1}} \cdot e^{\frac{1}{n+1}} \\ &= e^{-\frac{1}{n+1}} \end{aligned}$$

The Easy Case

Let $\gamma_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned}\gamma_n^2 &= \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1} \\ &= \left(1 - \frac{1}{n+1}\right)^2 \left(1 + \frac{1}{(n-1)(n+1)}\right)^{n-1} \\ &\leq e^{-2\frac{1}{n+1}} \cdot e^{\frac{1}{n+1}} \\ &= e^{-\frac{1}{n+1}}\end{aligned}$$

where we used $(1+x)^a \leq e^{ax}$ for $x \in \mathbb{R}$ and $a > 0$.

The Easy Case

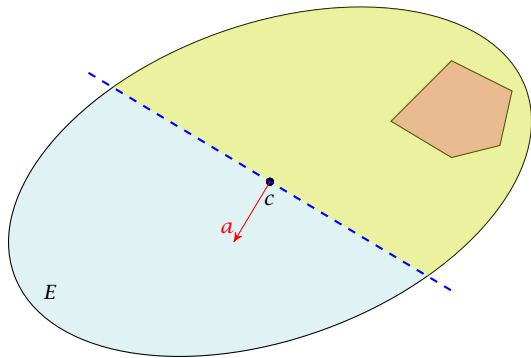
Let $y_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned} y_n^2 &= \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1} \\ &= \left(1 - \frac{1}{n+1}\right)^2 \left(1 + \frac{1}{(n-1)(n+1)}\right)^{n-1} \\ &\leq e^{-2\frac{1}{n+1}} \cdot e^{\frac{1}{n+1}} \\ &= e^{-\frac{1}{n+1}} \end{aligned}$$

where we used $(1+x)^a \leq e^{ax}$ for $x \in \mathbb{R}$ and $a > 0$.

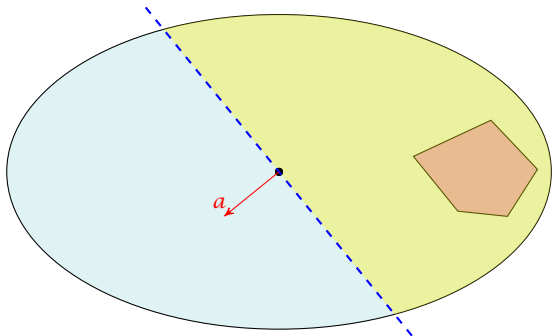
This gives $y_n \leq e^{-\frac{1}{2(n+1)}}$.

How to Compute the New Ellipsoid



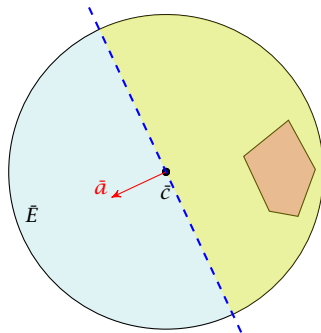
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.



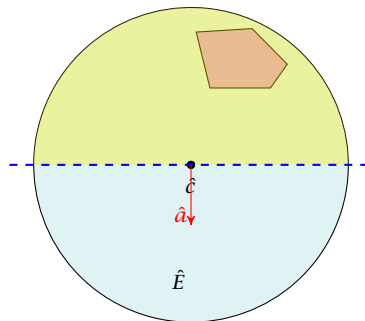
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.



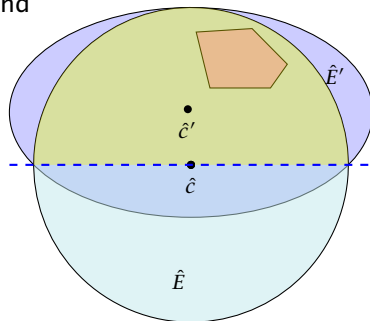
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .



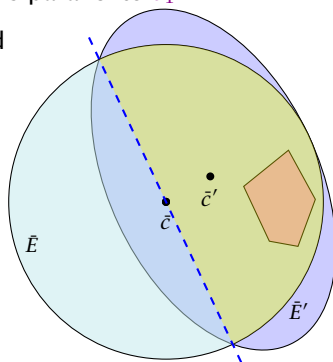
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.



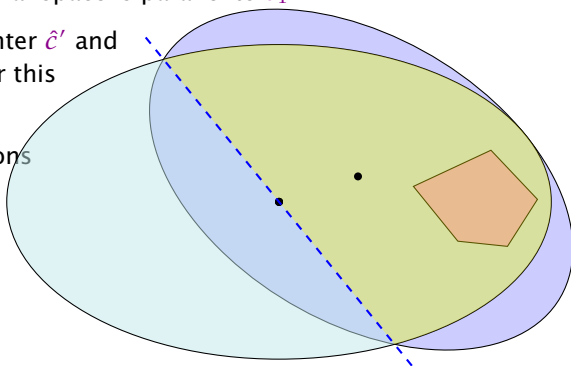
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\tilde{c}'$ and the new matrix $\tilde{Q}'$ for this simplified setting.
- ▶ Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E .



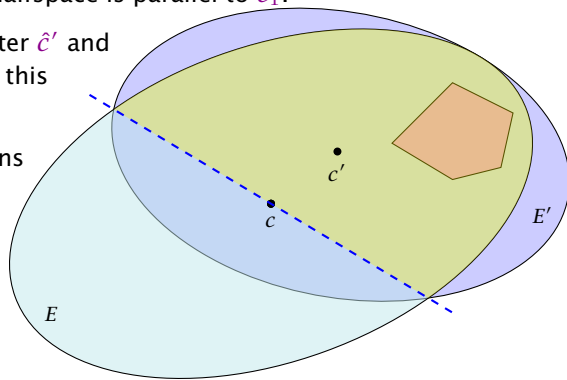
How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- ▶ Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E .



How to Compute the New Ellipsoid

- ▶ Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to translate/distort the ellipsoid (back) into the unit ball.
- ▶ Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1 .
- ▶ Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- ▶ Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E .



Our progress is the same:

$$e^{-\frac{1}{2(n+1)}}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0, 1))}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = \frac{\text{vol}(\hat{E}')}{\text{vol}(\hat{E})}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = \frac{\text{vol}(\hat{E}')}{\text{vol}(\hat{E})} = \frac{\text{vol}(R(\hat{E}'))}{\text{vol}(R(\hat{E}))}$$

Our progress is the same:

$$\begin{aligned} e^{-\frac{1}{2(n+1)}} &\geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = \frac{\text{vol}(\hat{E}')}{\text{vol}(\hat{E})} = \frac{\text{vol}(R(\hat{E}'))}{\text{vol}(R(\hat{E}))} \\ &= \frac{\text{vol}(\tilde{E}')}{\text{vol}(\tilde{E})} \end{aligned}$$

Our progress is the same:

$$\begin{aligned} e^{-\frac{1}{2(n+1)}} &\geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = \frac{\text{vol}(\hat{E}')}{\text{vol}(\hat{E})} = \frac{\text{vol}(R(\hat{E}'))}{\text{vol}(R(\hat{E}))} \\ &= \frac{\text{vol}(\tilde{E}')}{\text{vol}(\tilde{E})} = \frac{\text{vol}(f(\tilde{E}'))}{\text{vol}(f(\tilde{E}))} \end{aligned}$$

Our progress is the same:

$$\begin{aligned} e^{-\frac{1}{2(n+1)}} &\geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = \frac{\text{vol}(\hat{E}')}{\text{vol}(\hat{E})} = \frac{\text{vol}(R(\hat{E}'))}{\text{vol}(R(\hat{E}))} \\ &= \frac{\text{vol}(\tilde{E}')}{\text{vol}(\tilde{E})} = \frac{\text{vol}(f(\tilde{E}'))}{\text{vol}(f(\tilde{E}))} = \frac{\text{vol}(E')}{\text{vol}(E)} \end{aligned}$$

Our progress is the same:

$$\begin{aligned} e^{-\frac{1}{2(n+1)}} &\geq \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = \frac{\text{vol}(\hat{E}')}{\text{vol}(\hat{E})} = \frac{\text{vol}(R(\hat{E}'))}{\text{vol}(R(\hat{E}))} \\ &= \frac{\text{vol}(\tilde{E}')}{\text{vol}(\tilde{E})} = \frac{\text{vol}(f(\tilde{E}'))}{\text{vol}(f(\tilde{E}))} = \frac{\text{vol}(E')}{\text{vol}(E)} \end{aligned}$$

Here it is important that mapping a set with affine function $f(x) = Lx + t$ changes the volume by factor $\det(L)$.

The Ellipsoid Algorithm

How to compute the new parameters?

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

$$f^{-1}(H) = \{f^{-1}(x) \mid a^T(x - c) \leq 0\}$$

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^T(x - c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^T(f(y) - c) \leq 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^T(x - c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(f(y) - c) \leq 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^T(x - c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(Ly + c - c) \leq 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^T(x - c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(Ly + c - c) \leq 0\} \\ &= \{y \mid (a^T L)y \leq 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to compute the new parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^T(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^T(x - c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(f(y) - c) \leq 0\} \\ &= \{y \mid a^T(Ly + c - c) \leq 0\} \\ &= \{y \mid (a^T L)y \leq 0\} \end{aligned}$$

This means $\bar{a} = L^T a$.

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1} \left(\frac{L^T a}{\|L^T a\|} \right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1} \left(\frac{L^T a}{\|L^T a\|} \right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\bar{c}'$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1} \left(\frac{L^T a}{\|L^T a\|} \right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}'$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^T a}{\|L^T a\|}$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^T a}{\|L^T a\|}$$

$$c'$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\tilde{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^T a}{\|L^T a\|}$$

$$c' = f(\tilde{c}')$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\tilde{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^T a}{\|L^T a\|}$$

$$c' = f(\tilde{c}') = L \cdot \tilde{c}' + c$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\tilde{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^T a}{\|L^T a\|}$$

$$\begin{aligned} c' &= f(\tilde{c}') = L \cdot \tilde{c}' + c \\ &= -\frac{1}{n+1} L \frac{L^T a}{\|L^T a\|} + c \end{aligned}$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1 -direction. Hence,

$$R^{-1}\left(\frac{L^T a}{\|L^T a\|}\right) = -e_1 \quad \Rightarrow \quad -\frac{L^T a}{\|L^T a\|} = R \cdot e_1$$

Hence,

$$\tilde{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^T a}{\|L^T a\|}$$

$$\begin{aligned} c' &= f(\tilde{c}') = L \cdot \tilde{c}' + c \\ &= -\frac{1}{n+1} L \frac{L^T a}{\|L^T a\|} + c \\ &= c - \frac{1}{n+1} \frac{Qa}{\sqrt{a^T Qa}} \end{aligned}$$

For computing the matrix Q' of the new ellipsoid we assume in the following that $\hat{E}', \bar{E}'$ and E' refer to the ellipsoids centered in the origin.

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1} \left(I - \frac{2}{n+1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2-1$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2-1$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2 - 1$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2 - 1$

$$\begin{aligned} b^2 - b^2 \frac{2}{n + 1} &= \frac{n^2}{n^2 - 1} - \frac{2n^2}{(n - 1)(n + 1)^2} \\ &= \frac{n^2(n + 1) - 2n^2}{(n - 1)(n + 1)^2} = \frac{n^2(n - 1)}{(n - 1)(n + 1)^2} = a^2 \end{aligned}$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2 - 1$

$$\begin{aligned} b^2 - b^2 \frac{2}{n + 1} &= \frac{n^2}{n^2 - 1} - \frac{2n^2}{(n - 1)(n + 1)^2} \\ &= \frac{n^2(n + 1) - 2n^2}{(n - 1)(n + 1)^2} = \frac{n^2(n - 1)}{(n - 1)(n + 1)^2} = a^2 \end{aligned}$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2 - 1$

$$\begin{aligned} b^2 - b^2 \frac{2}{n + 1} &= \frac{n^2}{n^2 - 1} - \frac{2n^2}{(n - 1)(n + 1)^2} \\ &= \frac{n^2(n + 1) - 2n^2}{(n - 1)(n + 1)^2} = \frac{n^2(n - 1)}{(n - 1)(n + 1)^2} = a^2 \end{aligned}$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2 - 1$

$$\begin{aligned} b^2 - b^2 \frac{2}{n + 1} &= \frac{n^2}{n^2 - 1} - \frac{2n^2}{(n - 1)(n + 1)^2} \\ &= \frac{n^2(n + 1) - 2n^2}{(n - 1)(n + 1)^2} = \frac{n^2(n - 1)}{(n - 1)(n + 1)^2} = a^2 \end{aligned}$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right)$$

because for $a^2 = n^2/(n+1)^2$ and $b^2 = n^2/n^2-1$

$$\begin{aligned} b^2 - b^2 \frac{2}{n+1} &= \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2} \\ &= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2 \end{aligned}$$

9 The Ellipsoid Algorithm

$$\bar{E}'$$

9 The Ellipsoid Algorithm

$$\bar{E}' = R(\hat{E}')$$

9 The Ellipsoid Algorithm

$$\begin{aligned}\bar{E}' &= R(\hat{E}') \\ &= \{R(x) \mid x^T \hat{Q}'^{-1} x \leq 1\}\end{aligned}$$

9 The Ellipsoid Algorithm

$$\begin{aligned}\bar{E}' &= R(\hat{E}') \\ &= \{R(x) \mid x^T \hat{Q}'^{-1} x \leq 1\} \\ &= \{y \mid (R^{-1}y)^T \hat{Q}'^{-1} R^{-1}y \leq 1\}\end{aligned}$$

9 The Ellipsoid Algorithm

$$\begin{aligned}\bar{E}' &= R(\hat{E}') \\ &= \{R(x) \mid x^T \hat{Q}'^{-1} x \leq 1\} \\ &= \{y \mid (R^{-1}y)^T \hat{Q}'^{-1} R^{-1}y \leq 1\} \\ &= \{y \mid y^T (R^T)^{-1} \hat{Q}'^{-1} R^{-1}y \leq 1\}\end{aligned}$$

9 The Ellipsoid Algorithm

$$\begin{aligned}\bar{E}' &= R(\hat{E}') \\ &= \{R(x) \mid x^T \hat{Q}'^{-1} x \leq 1\} \\ &= \{y \mid (R^{-1}y)^T \hat{Q}'^{-1} R^{-1}y \leq 1\} \\ &= \{y \mid y^T (R^T)^{-1} \hat{Q}'^{-1} R^{-1}y \leq 1\} \\ &= \{y \mid y^T \underbrace{(R \hat{Q}' R^T)^{-1}}_{\hat{Q}'} y \leq 1\}\end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

$$\bar{Q}'$$

9 The Ellipsoid Algorithm

Hence,

$$\bar{Q}' = R\hat{Q}'R^T$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned}\bar{Q}' &= R\hat{Q}'R^T \\ &= R \cdot \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n + 1} e_1 e_1^T \right) \cdot R^T\end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned}\bar{Q}' &= R\hat{Q}'R^T \\ &= R \cdot \frac{n^2}{n^2-1} \left(I - \frac{2}{n+1} e_1 e_1^T \right) \cdot R^T \\ &= \frac{n^2}{n^2-1} \left(R \cdot R^T - \frac{2}{n+1} (Re_1)(Re_1)^T \right)\end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned}\bar{Q}' &= R\hat{Q}'R^T \\ &= R \cdot \frac{n^2}{n^2-1} \left(I - \frac{2}{n+1} e_1 e_1^T \right) \cdot R^T \\ &= \frac{n^2}{n^2-1} \left(R \cdot R^T - \frac{2}{n+1} (Re_1)(Re_1)^T \right) \\ &= \frac{n^2}{n^2-1} \left(I - \frac{2}{n+1} \frac{L^T a a^T L}{\|L^T a\|^2} \right)\end{aligned}$$

9 The Ellipsoid Algorithm

E'

9 The Ellipsoid Algorithm

$$E' = L(\bar{E}')$$

9 The Ellipsoid Algorithm

$$\begin{aligned} E' &= L(\tilde{E}') \\ &= \{L(x) \mid x^T \tilde{Q}'^{-1} x \leq 1\} \end{aligned}$$

9 The Ellipsoid Algorithm

$$\begin{aligned} E' &= L(\tilde{E}') \\ &= \{L(x) \mid x^T \tilde{Q}'^{-1} x \leq 1\} \\ &= \{y \mid (L^{-1}y)^T \tilde{Q}'^{-1} L^{-1}y \leq 1\} \end{aligned}$$

9 The Ellipsoid Algorithm

$$\begin{aligned} E' &= L(\tilde{E}') \\ &= \{L(x) \mid x^T \tilde{Q}'^{-1} x \leq 1\} \\ &= \{y \mid (L^{-1}y)^T \tilde{Q}'^{-1} L^{-1}y \leq 1\} \\ &= \{y \mid y^T (L^T)^{-1} \tilde{Q}'^{-1} L^{-1}y \leq 1\} \end{aligned}$$

9 The Ellipsoid Algorithm

$$\begin{aligned} E' &= L(\tilde{E}') \\ &= \{L(x) \mid x^T \tilde{Q}'^{-1} x \leq 1\} \\ &= \{y \mid (L^{-1}y)^T \tilde{Q}'^{-1} L^{-1}y \leq 1\} \\ &= \{y \mid y^T (L^T)^{-1} \tilde{Q}'^{-1} L^{-1}y \leq 1\} \\ &= \{y \mid y^T \underbrace{(L\tilde{Q}'L^T)^{-1}}_{Q'} y \leq 1\} \end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

Q'

9 The Ellipsoid Algorithm

Hence,

$$Q' = L\bar{Q}'L^T$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned} Q' &= L\bar{Q}'L^T \\ &= L \cdot \frac{n^2}{n^2 - 1} \left(I - \frac{2}{n+1} \frac{L^T a a^T L}{a^T Q a} \right) \cdot L^T \end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned} Q' &= L\bar{Q}'L^T \\ &= L \cdot \frac{n^2}{n^2-1} \left(I - \frac{2}{n+1} \frac{L^T a a^T L}{a^T Q a} \right) \cdot L^T \\ &= \frac{n^2}{n^2-1} \left(Q - \frac{2}{n+1} \frac{Q a a^T Q}{a^T Q a} \right) \end{aligned}$$

Incomplete Algorithm

Algorithm 1 ellipsoid-algorithm

```
1: input: point  $c \in \mathbb{R}^n$ , convex set  $K \subseteq \mathbb{R}^n$ 
2: output: point  $x \in K$  or “ $K$  is empty”
3:  $Q \leftarrow ???$ 
4: repeat
5:     if  $c \in K$  then return  $c$ 
6:     else
7:         choose a violated hyperplane  $a$ 
8:         
$$c \leftarrow c - \frac{1}{n+1} \frac{Qa}{\sqrt{a^T Q a}}$$

9:         
$$Q \leftarrow \frac{n^2}{n^2 - 1} \left( Q - \frac{2}{n+1} \frac{Qaa^T Q}{a^T Q a} \right)$$

10:    endif
11: until  $???$ 
12: return “ $K$  is empty”
```

Repeat: Size of basic solutions

Lemma 52

Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a bounded polyhedron. Let $L := 2\langle A \rangle + \langle b \rangle + 2n(1 + \log_2 n)$. Then every entry x_j in a basic solution fulfills $|x_j| = \frac{D_j}{D}$ with $D_j, D \leq 2^L$.

In the following we use $\delta := 2^L$.

Proof:

We can replace P by $P' := \{x \mid A'x \leq b; x \geq 0\}$ where $A' = \begin{bmatrix} A \\ -A \end{bmatrix}$. The lemma follows by applying Lemma 47, and observing that $\langle A' \rangle = 2\langle A \rangle$ and $n' = 2n$.

Repeat: Size of basic solutions

Lemma 52

Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a bounded polyhedron. Let $L := 2\langle A \rangle + \langle b \rangle + 2n(1 + \log_2 n)$. Then every entry x_j in a basic solution fulfills $|x_j| = \frac{D_j}{D}$ with $D_j, D \leq 2^L$.

In the following we use $\delta := 2^L$.

Proof:

We can replace P by $P' := \{x \mid A'x \leq b; x \geq 0\}$ where $A' = \begin{bmatrix} A \\ -A \end{bmatrix}$. The lemma follows by applying Lemma 47, and observing that $\langle A' \rangle = 2\langle A \rangle$ and $n' = 2n$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n \text{vol}(B(0, 1)) \leq (n\delta)^n \text{vol}(B(0, 1))$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n \text{vol}(B(0, 1)) \leq (n\delta)^n \text{vol}(B(0, 1))$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n \text{vol}(B(0, 1)) \leq (n\delta)^n \text{vol}(B(0, 1))$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n \text{vol}(B(0, 1)) \leq (n\delta)^n \text{vol}(B(0, 1))$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n \text{vol}(B(0, 1)) \leq (n\delta)^n \text{vol}(B(0, 1))$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n \text{vol}(B(0, 1)) \leq (n\delta)^n \text{vol}(B(0, 1))$.

When can we terminate?

Let $P := \{x \mid Ax \leq b\}$ with $A \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be a bounded polytop.

Consider the following polyhedron

$$P_\lambda := \left\{ x \mid Ax \leq b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\},$$

where $\lambda = \delta^2 + 1$.

Note that the volume of P_λ cannot be 0

When can we terminate?

Let $P := \{x \mid Ax \leq b\}$ with $A \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be a bounded polytop.

Consider the following polyhedron

$$P_\lambda := \left\{ x \mid Ax \leq b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\},$$

where $\lambda = \delta^2 + 1$.

Note that the volume of P_λ cannot be 0

When can we terminate?

Let $P := \{x \mid Ax \leq b\}$ with $A \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be a bounded polytop.

Consider the following polyhedron

$$P_\lambda := \left\{ x \mid Ax \leq b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\},$$

where $\lambda = \delta^2 + 1$.

Note that the volume of P_λ cannot be 0

Making P full-dimensional

Lemma 53

P_λ is feasible if and only if P is feasible.

$\Leftarrow$: obvious!

Making P full-dimensional

Lemma 53

P_λ is feasible if and only if P is feasible.

$\Leftarrow$: obvious!

Making P full-dimensional

$\Rightarrow$:

Consider the polyhedrons

$$\bar{P} = \{x \mid [A \ -A \ I_m]x = b; x \geq 0\}$$

and

$$\bar{P}_\lambda = \left\{x \mid [A \ -A \ I_m]x = b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}; x \geq 0\right\}.$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

Making P full-dimensional

$\Rightarrow$:

Consider the polyhedrons

$$\bar{P} = \{x \mid [A \ -A \ I_m]x = b; x \geq 0\}$$

and

$$\bar{P}_\lambda = \left\{x \mid [A \ -A \ I_m]x = b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}; x \geq 0\right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

Making P full-dimensional

$\Rightarrow$:

Consider the polyhedrons

$$\bar{P} = \{x \mid [A \ -A \ I_m]x = b; x \geq 0\}$$

and

$$\bar{P}_\lambda = \left\{x \mid [A \ -A \ I_m]x = b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}; x \geq 0\right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

Making P full-dimensional

$\Rightarrow$:

Consider the polyhedrons

$$\bar{P} = \{x \mid [A \ -A \ I_m]x = b; x \geq 0\}$$

and

$$\bar{P}_\lambda = \left\{x \mid [A \ -A \ I_m]x = b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}; x \geq 0\right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

Making P full-dimensional

Let $\bar{A} = \begin{bmatrix} A & -A & I_m \end{bmatrix}$.

$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}b + \frac{1}{\lambda}\bar{A}_B^{-1}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

(The other x -values are zero)

The only reason that this basic feasible solution is not feasible for $\bar{P}$ is that one of the basic variables becomes negative.

Hence, there exists i with

$$(\bar{A}_B^{-1}b)_i < 0 \leq (\bar{A}_B^{-1}b)_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\bar{1})_i$$

Making P full-dimensional

Let $\bar{A} = \begin{bmatrix} A & -A & I_m \end{bmatrix}$.

$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}b + \frac{1}{\lambda}\bar{A}_B^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

(The other x -values are zero)

The only reason that this basic feasible solution is not feasible for $\bar{P}$ is that one of the basic variables becomes negative.

Hence, there exists i with

$$(\bar{A}_B^{-1}b)_i < 0 \leq (\bar{A}_B^{-1}b)_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\bar{1})_i$$

Making P full-dimensional

Let $\bar{A} = \begin{bmatrix} A & -A & I_m \end{bmatrix}$.

$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}b + \frac{1}{\lambda}\bar{A}_B^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

(The other x -values are zero)

The only reason that this basic feasible solution is not feasible for $\bar{P}$ is that one of the basic variables becomes negative.

Hence, there exists i with

$$(\bar{A}_B^{-1}b)_i < 0 \leq (\bar{A}_B^{-1}b)_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i$$

Making P full-dimensional

By Cramers rule we get

$$(\bar{A}_B^{-1}b)_i < 0 \quad \Rightarrow \quad (\bar{A}_B^{-1}b)_i \leq -\frac{1}{\det(\bar{A}_B)} \leq -1/\delta$$

and

$$(\bar{A}_B^{-1}\vec{1})_i \leq \det(\bar{A}_B^j) \leq \delta ,$$

where $\bar{A}_B^j$ is obtained by replacing the j -th column of $\bar{A}_B$ by $\vec{1}$.

But then

$$(\bar{A}_B^{-1}b)_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i \leq -1/\delta + \delta/\lambda < 0 ,$$

as we chose $\lambda = \delta^2 + 1$. Contradiction.

Making P full-dimensional

By Cramers rule we get

$$(\bar{A}_B^{-1}b)_i < 0 \quad \Rightarrow \quad (\bar{A}_B^{-1}b)_i \leq -\frac{1}{\det(\bar{A}_B)} \leq -1/\delta$$

and

$$(\bar{A}_B^{-1}\vec{1})_i \leq \det(\bar{A}_B^j) \leq \delta ,$$

where $\bar{A}_B^j$ is obtained by replacing the j -th column of $\bar{A}_B$ by $\vec{1}$.

But then

$$(\bar{A}_B^{-1}b)_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i \leq -1/\delta + \delta/\lambda < 0 ,$$

as we chose $\lambda = \delta^2 + 1$. Contradiction.

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$(A(x + \vec{\ell}))_i$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$(A(x + \vec{\ell}))_i = (Ax)_i + (A\vec{\ell})_i$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$(A(x + \vec{\ell}))_i = (Ax)_i + (A\vec{\ell})_i \leq b_i + \vec{a}_i^T \vec{\ell}$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$\begin{aligned}(A(x + \vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \leq b_i + \vec{a}_i^T \vec{\ell} \\ &\leq b_i + \|\vec{a}_i\| \cdot \|\vec{\ell}\|\end{aligned}$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$\begin{aligned}(A(x + \vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \leq b_i + \vec{a}_i^T \vec{\ell} \\ &\leq b_i + \|\vec{a}_i\| \cdot \|\vec{\ell}\| \leq b_i + \sqrt{n} \cdot 2^{\langle a_{\max} \rangle} \cdot r\end{aligned}$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$\begin{aligned}(A(x + \vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \leq b_i + \vec{a}_i^T \vec{\ell} \\ &\leq b_i + \|\vec{a}_i\| \cdot \|\vec{\ell}\| \leq b_i + \sqrt{n} \cdot 2^{\langle a_{\max} \rangle} \cdot r \\ &\leq b_i + \frac{\sqrt{n} \cdot 2^{\langle a_{\max} \rangle}}{\delta^3}\end{aligned}$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$\begin{aligned}(A(x + \vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \leq b_i + \vec{a}_i^T \vec{\ell} \\ &\leq b_i + \|\vec{a}_i\| \cdot \|\vec{\ell}\| \leq b_i + \sqrt{n} \cdot 2^{\langle a_{\max} \rangle} \cdot r \\ &\leq b_i + \frac{\sqrt{n} \cdot 2^{\langle a_{\max} \rangle}}{\delta^3} \leq b_i + \frac{1}{\delta^2 + 1} \leq b_i + \frac{1}{\lambda}\end{aligned}$$

Lemma 54

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \text{vol}(B(0, 1)) = \frac{1}{\delta^{3n}} \text{vol}(B(0, 1))$.

Proof:

If P_λ feasible then also P . Let x be feasible for P .

This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$\begin{aligned}(A(x + \vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \leq b_i + \vec{a}_i^T \vec{\ell} \\ &\leq b_i + \|\vec{a}_i\| \cdot \|\vec{\ell}\| \leq b_i + \sqrt{n} \cdot 2^{\langle a_{\max} \rangle} \cdot r \\ &\leq b_i + \frac{\sqrt{n} \cdot 2^{\langle a_{\max} \rangle}}{\delta^3} \leq b_i + \frac{1}{\delta^2 + 1} \leq b_i + \frac{1}{\lambda}\end{aligned}$$

Hence, $x + \vec{\ell}$ is feasible for P_λ which proves the lemma.

How many iterations do we need until the volume becomes too small?

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \text{vol}(B(0, R)) < \text{vol}(B(0, r))$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \text{vol}(B(0, R)) < \text{vol}(B(0, r))$$

Hence,

i

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \text{vol}(B(0, R)) < \text{vol}(B(0, r))$$

Hence,

$$i > 2(n+1) \ln \left(\frac{\text{vol}(B(0, R))}{\text{vol}(B(0, r))} \right)$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \text{vol}(B(0, R)) < \text{vol}(B(0, r))$$

Hence,

$$\begin{aligned} i &> 2(n+1) \ln \left(\frac{\text{vol}(B(0, R))}{\text{vol}(B(0, r))} \right) \\ &= 2(n+1) \ln \left(n^n \delta^n \cdot \delta^{3n} \right) \end{aligned}$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \text{vol}(B(0, R)) < \text{vol}(B(0, r))$$

Hence,

$$\begin{aligned} i &> 2(n+1) \ln \left(\frac{\text{vol}(B(0, R))}{\text{vol}(B(0, r))} \right) \\ &= 2(n+1) \ln \left(n^n \delta^n \cdot \delta^{3n} \right) \\ &= 8n(n+1) \ln(\delta) + 2(n+1)n \ln(n) \end{aligned}$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \text{vol}(B(0, R)) < \text{vol}(B(0, r))$$

Hence,

$$\begin{aligned} i &> 2(n+1) \ln \left(\frac{\text{vol}(B(0, R))}{\text{vol}(B(0, r))} \right) \\ &= 2(n+1) \ln \left(n^n \delta^n \cdot \delta^{3n} \right) \\ &= 8n(n+1) \ln(\delta) + 2(n+1)n \ln(n) \\ &= \mathcal{O}(\text{poly}(n) \cdot L) \end{aligned}$$

Algorithm 1 ellipsoid-algorithm

```
1: input: point  $c \in \mathbb{R}^n$ , convex set  $K \subseteq \mathbb{R}^n$ , radii  $R$  and  $r$   
2:         with  $K \subseteq B(c, R)$ , and  $B(x, r) \subseteq K$  for some  $x$   
3: output: point  $x \in K$  or “ $K$  is empty”  
4:  $Q \leftarrow \text{diag}(R^2, \dots, R^2)$  // i.e.,  $L = \text{diag}(R, \dots, R)$   
5: repeat  
6:     if  $c \in K$  then return  $c$   
7:     else  
8:         choose a violated hyperplane  $a$   
9:         
$$c \leftarrow c - \frac{1}{n+1} \frac{Qa}{\sqrt{a^T Q a}}$$
  
10:        
$$Q \leftarrow \frac{n^2}{n^2 - 1} \left( Q - \frac{2}{n+1} \frac{Q a a^T Q}{a^T Q a} \right)$$
  
11:     endif  
12: until  $\det(Q) \leq r^{2n}$  // i.e.,  $\det(L) \leq r^n$   
13: return “ $K$  is empty”
```

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- ▶ a guarantee that a ball of radius r is contained in K ,
- ▶ an initial ball $B(x, R)$ with radius R that contains K ,
- ▶ a separation oracle for K .

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations.
Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- ▶ a guarantee that a ball of radius r is contained in K ,
- ▶ an initial ball B of radius R that contains K ,
- ▶ a separation oracle.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations.
Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- ▶ a guarantee that a ball of radius r is contained in K (inner ball)
- ▶ an initial ball B of radius R that contains K (outer ball)
- ▶ a separation oracle.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations.
Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- ▶ a guarantee that a ball of radius r is contained in K ,
- ▶ an initial ball $B(c, R)$ with radius R that contains K ,
- ▶ a separation oracle for K .

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations.
Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- ▶ a guarantee that a ball of radius r is contained in K ,
- ▶ an initial ball $B(c, R)$ with radius R that contains K ,
- ▶ a separation oracle for K .

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations.
Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- ▶ a guarantee that a ball of radius r is contained in K ,
- ▶ an initial ball $B(c, R)$ with radius R that contains K ,
- ▶ a separation oracle for K .

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations.
Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- ▶ certifies that $x \in K$,
- ▶ or finds a hyperplane separating x from K .

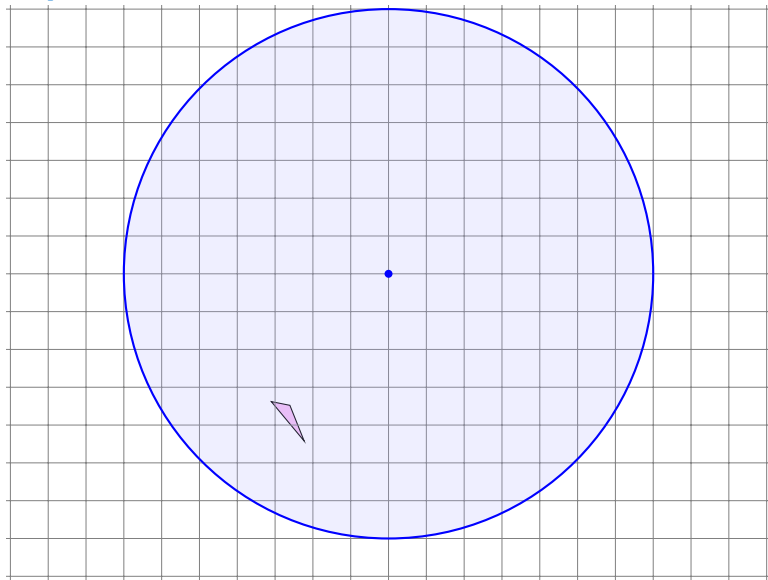
We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

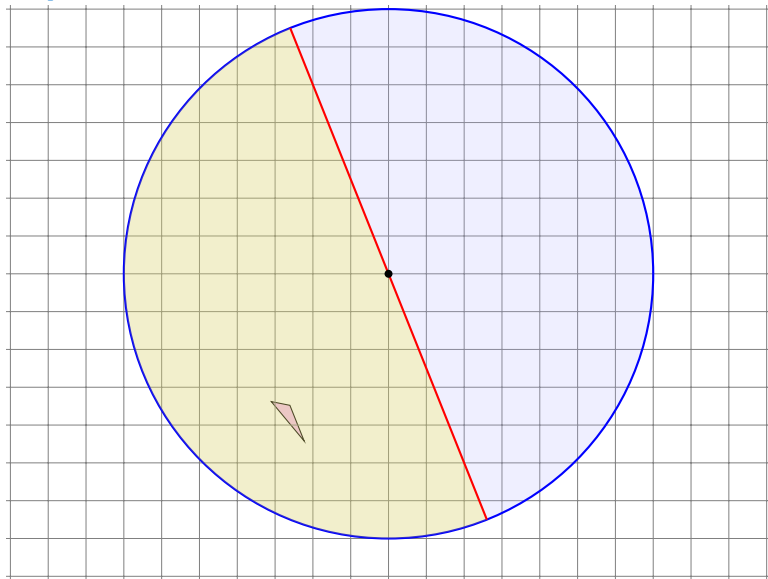
- ▶ a guarantee that a ball of radius r is contained in K ,
- ▶ an initial ball $B(c, R)$ with radius R that contains K ,
- ▶ a separation oracle for K .

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

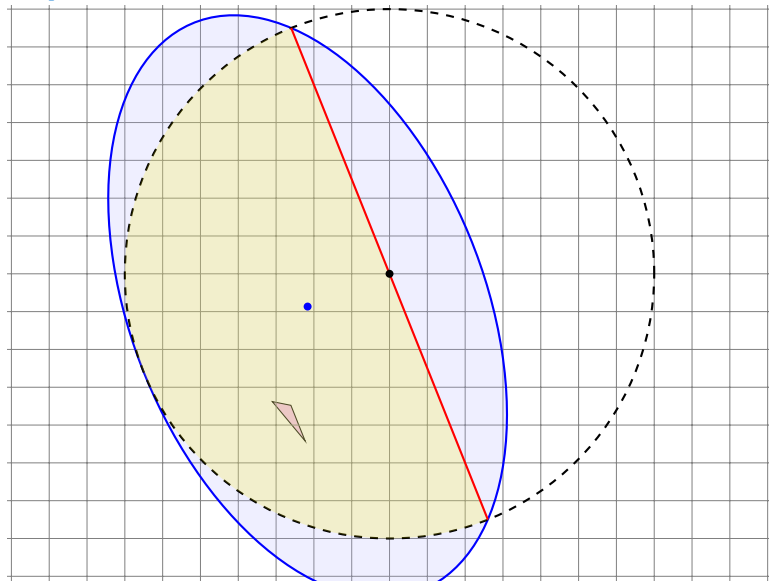
Example



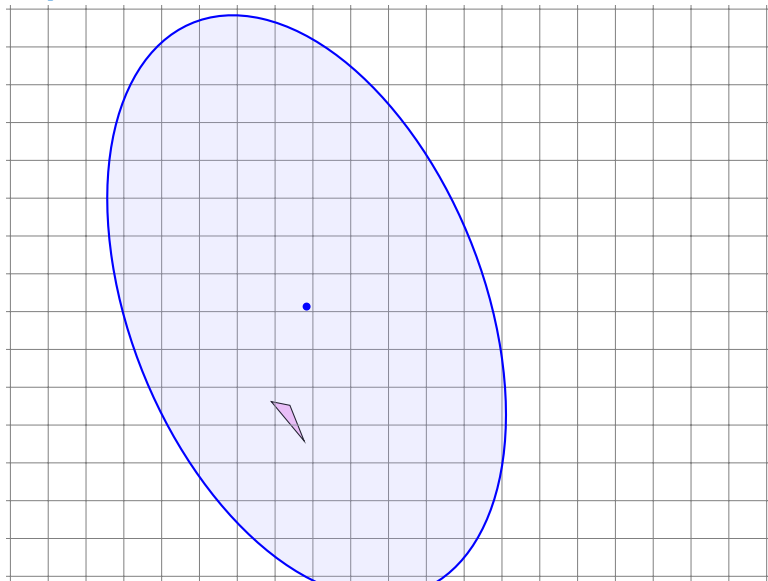
Example



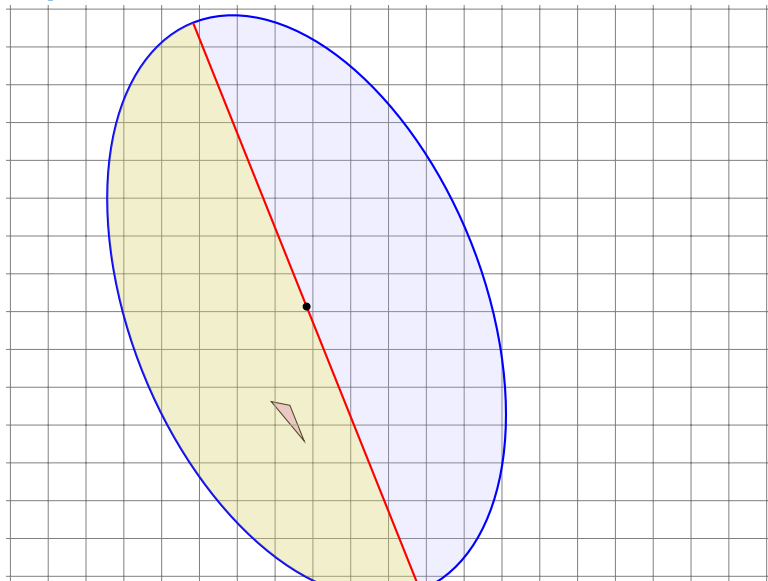
Example



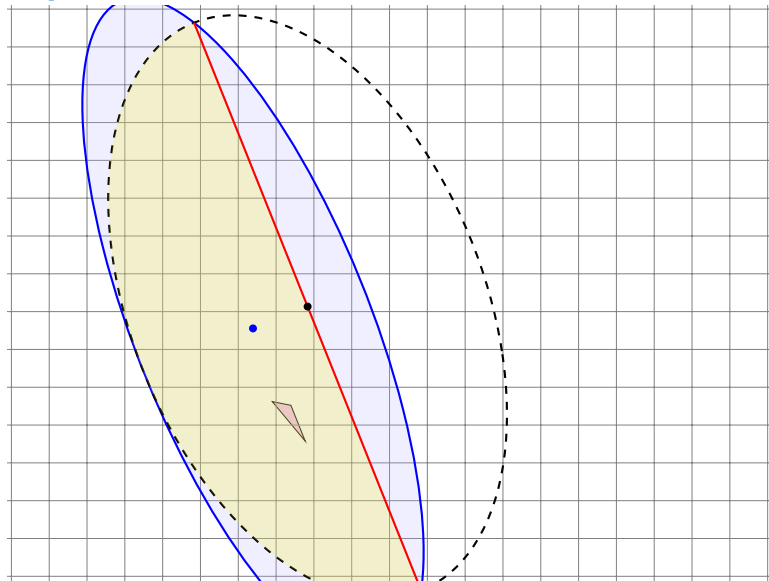
Example



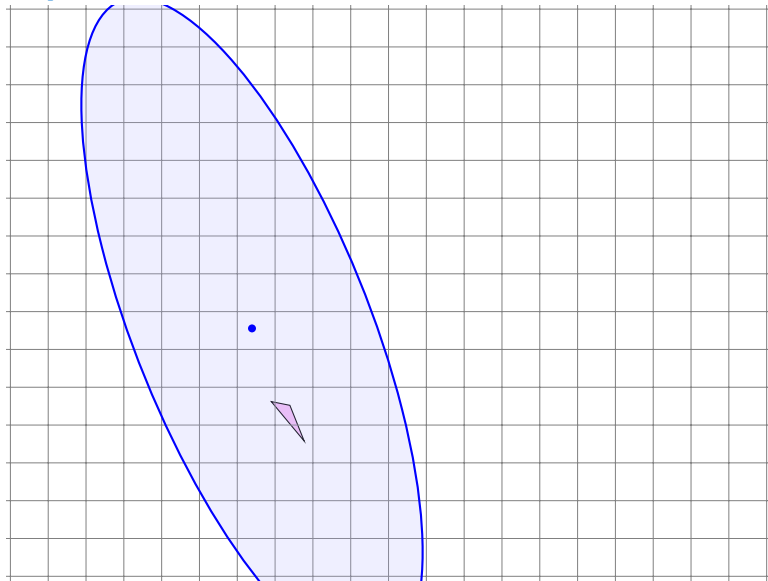
Example



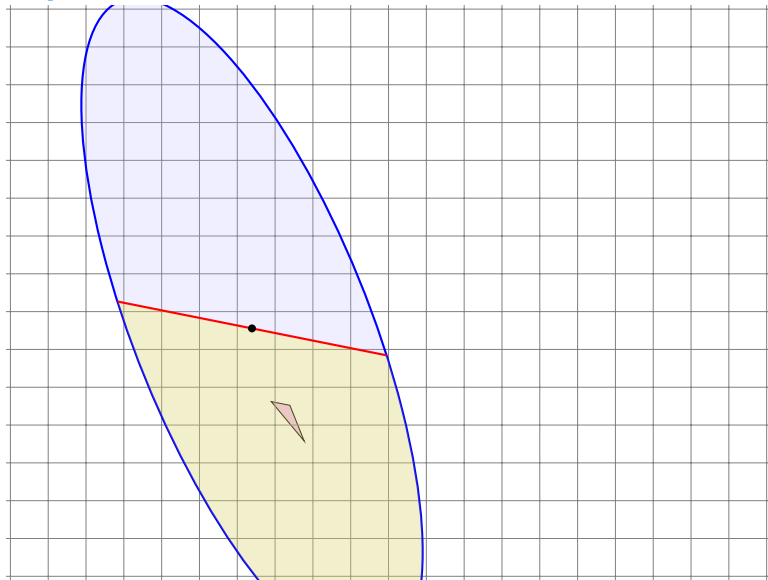
Example



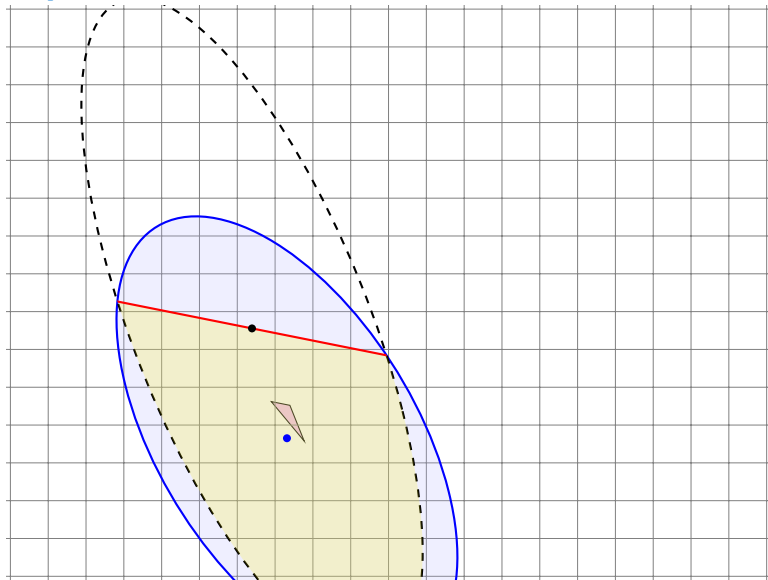
Example



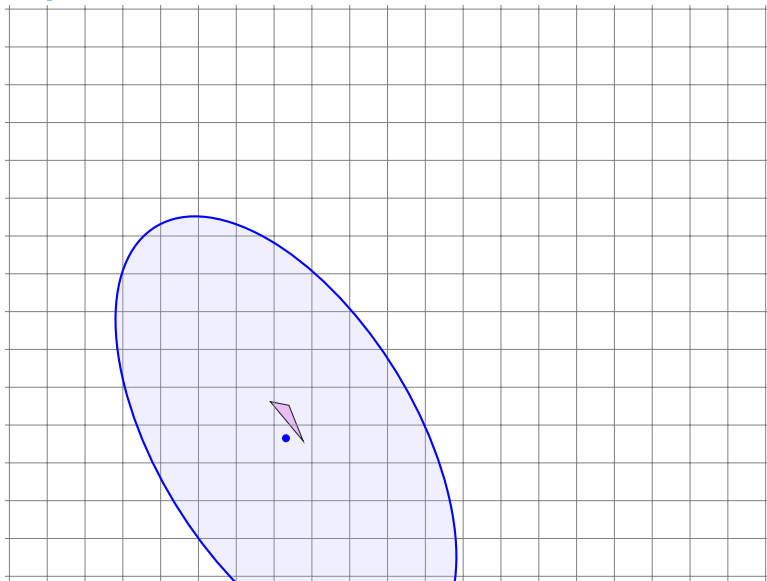
Example



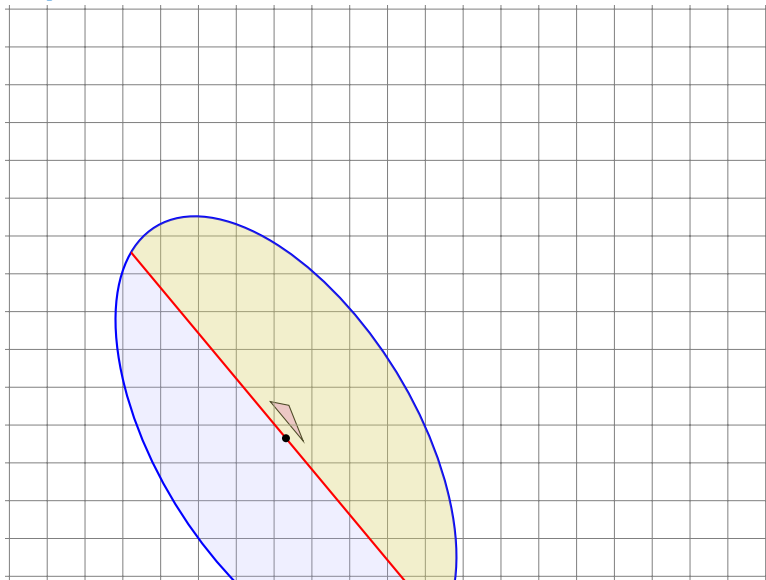
Example



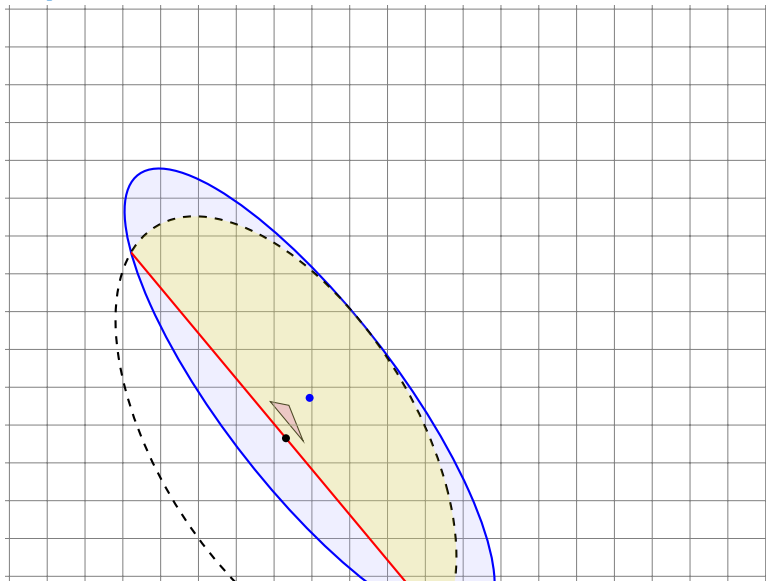
Example



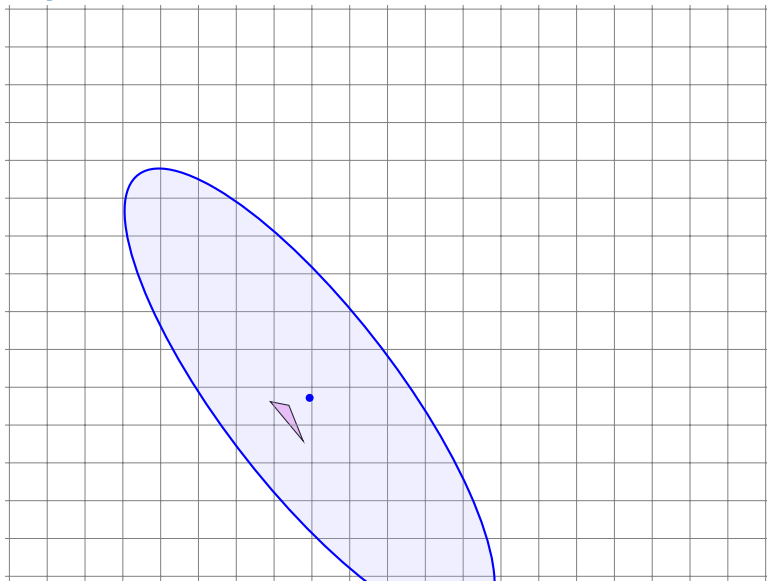
Example



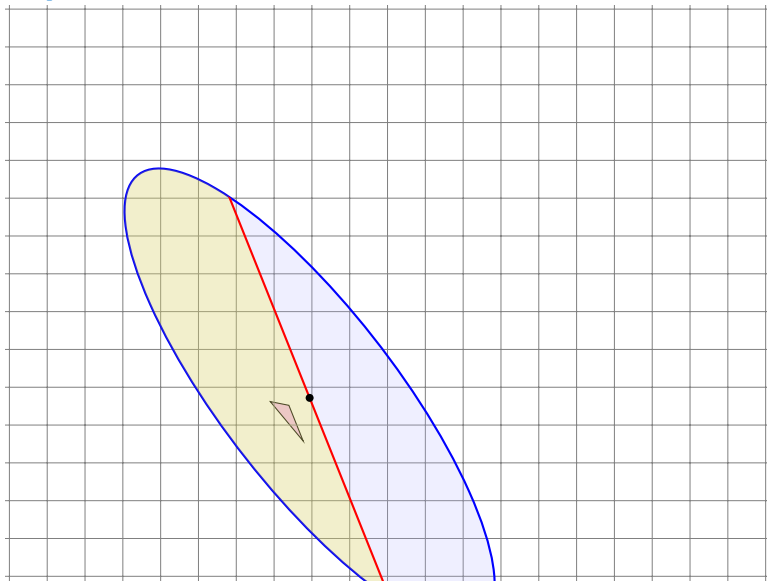
Example



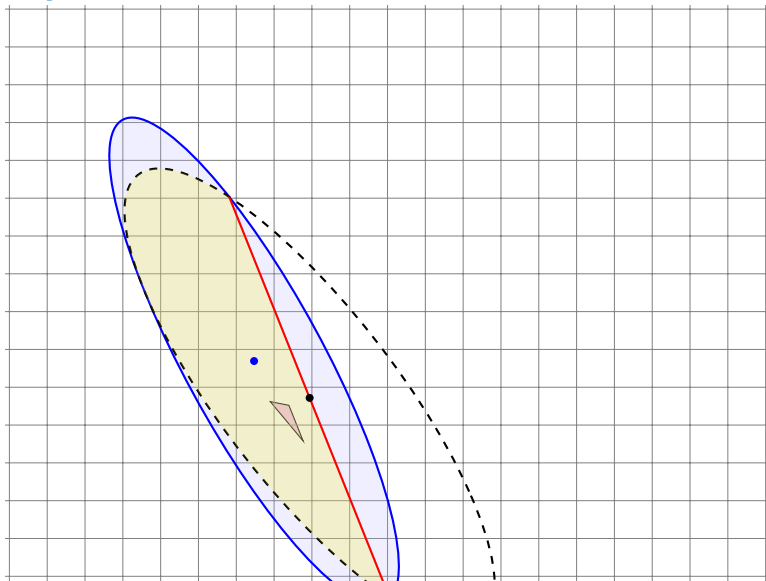
Example



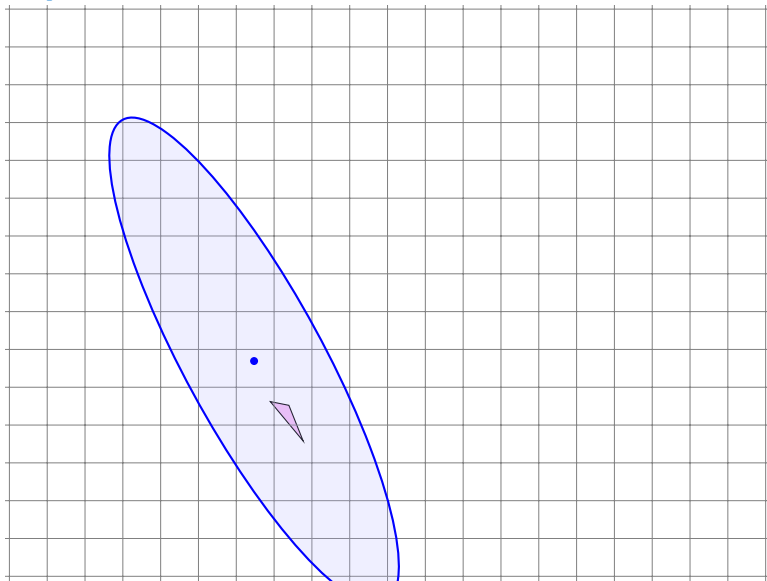
Example



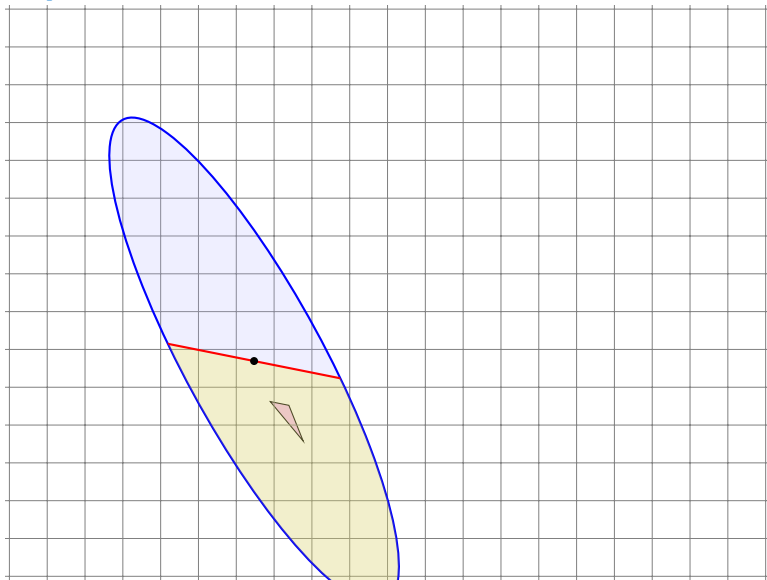
Example



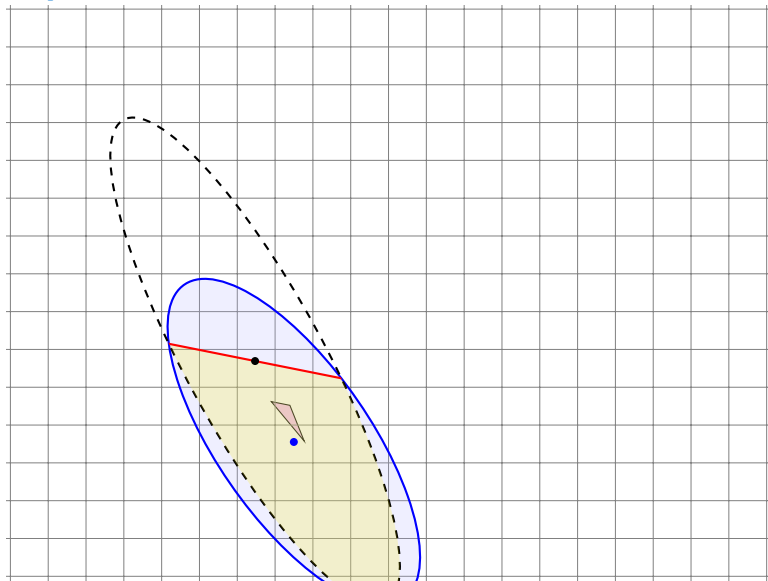
Example



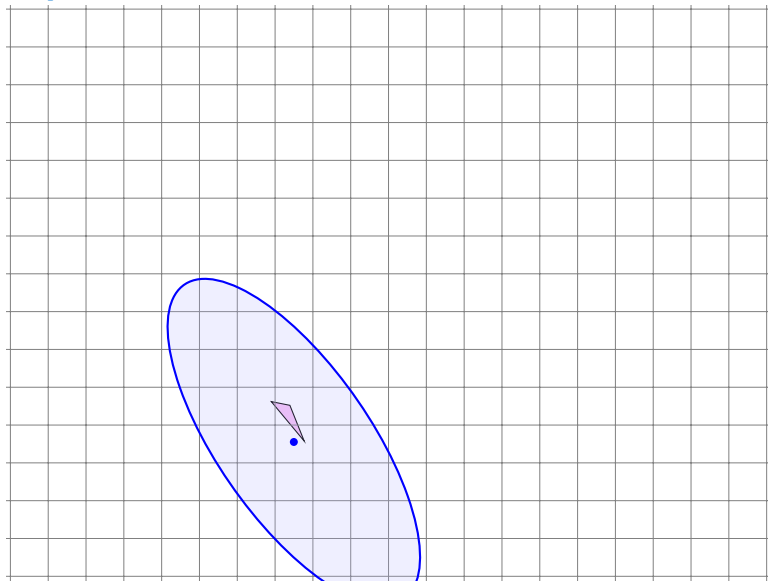
Example



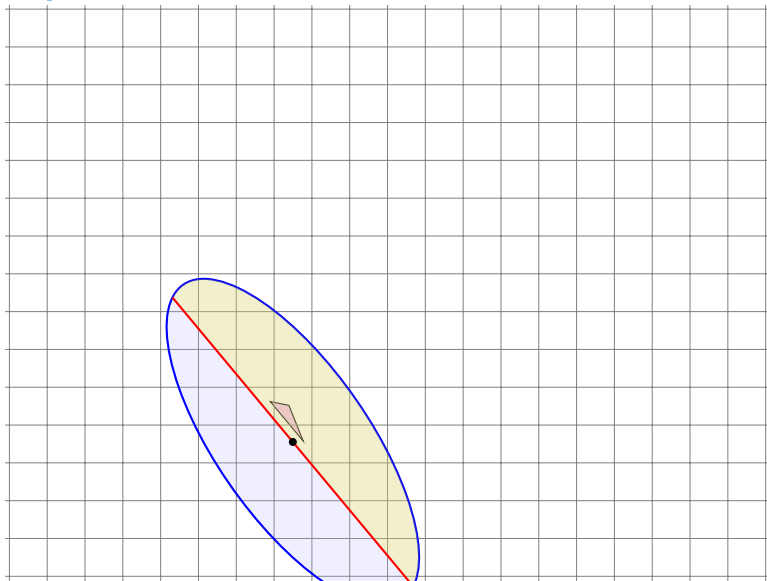
Example



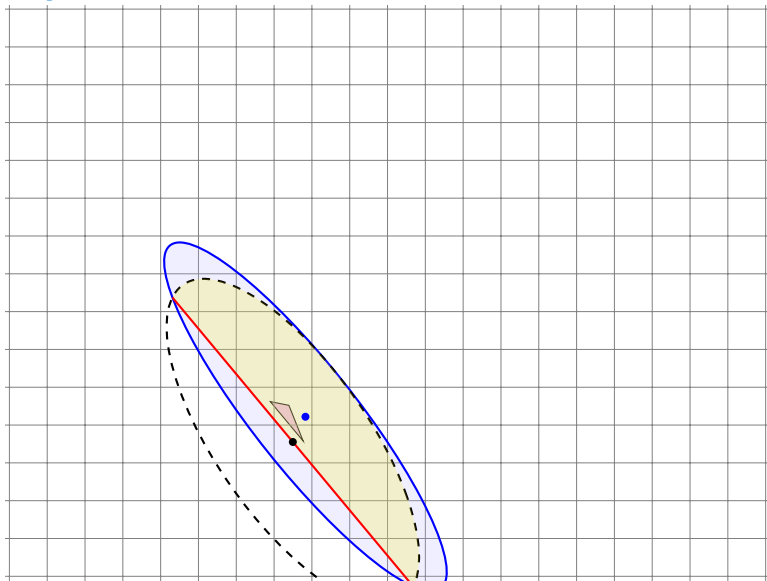
Example



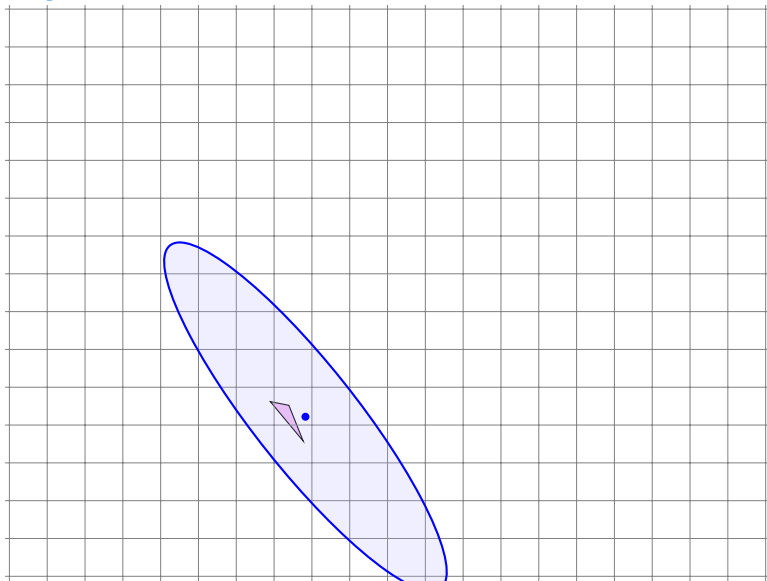
Example



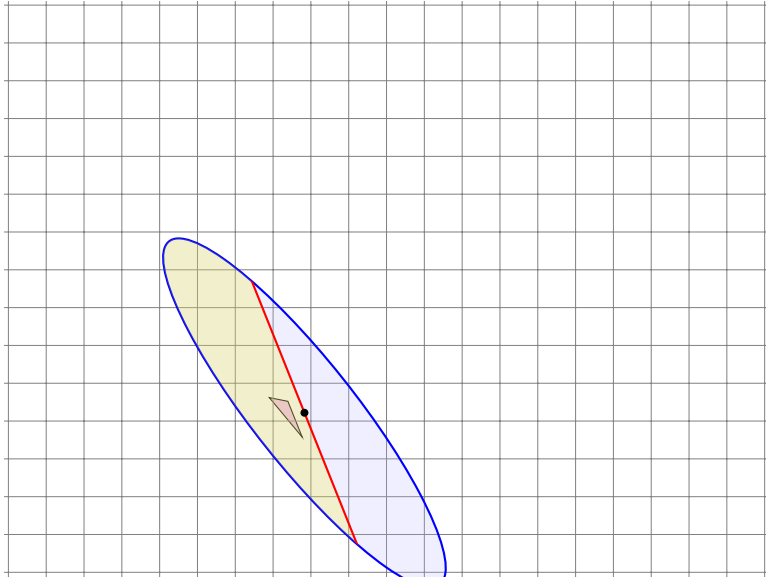
Example



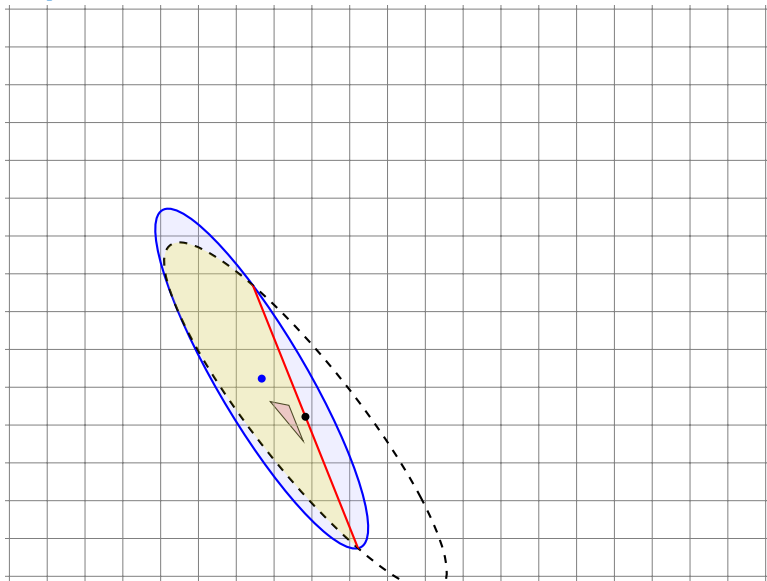
Example



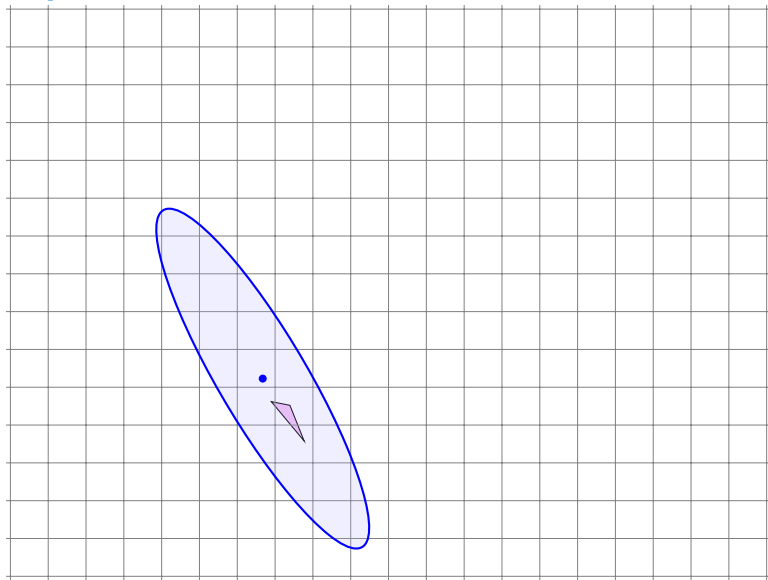
Example



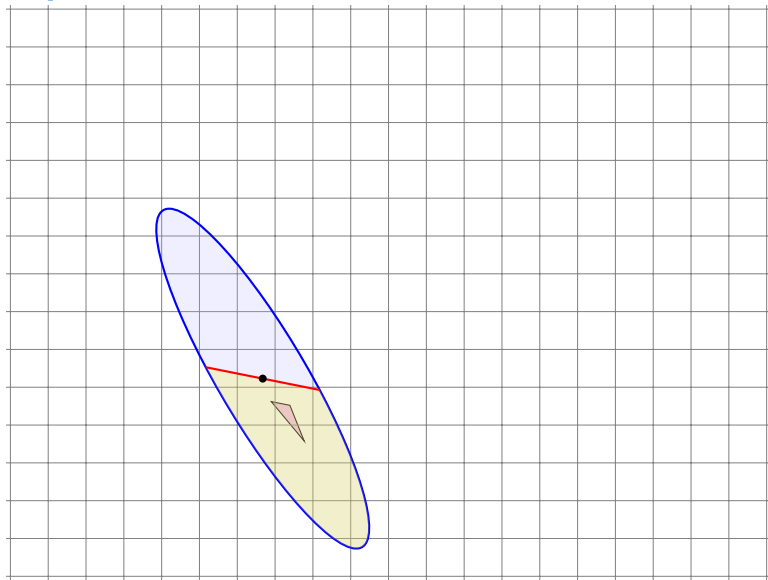
Example



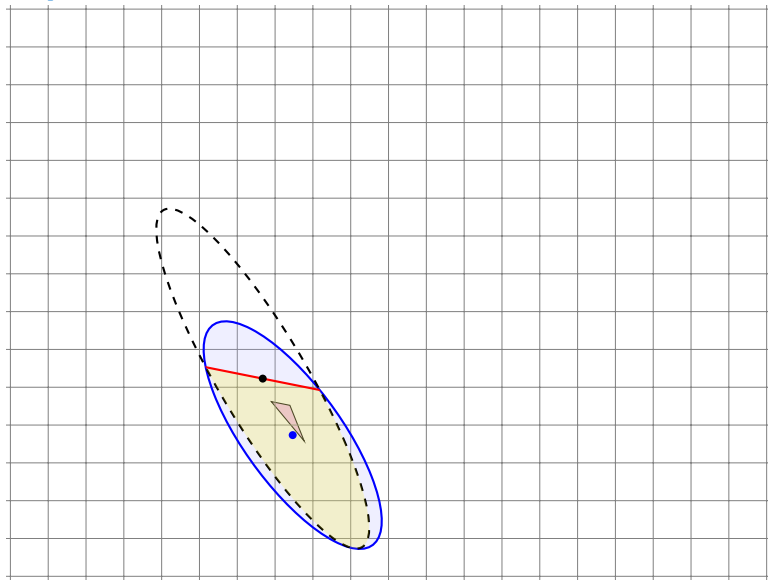
Example



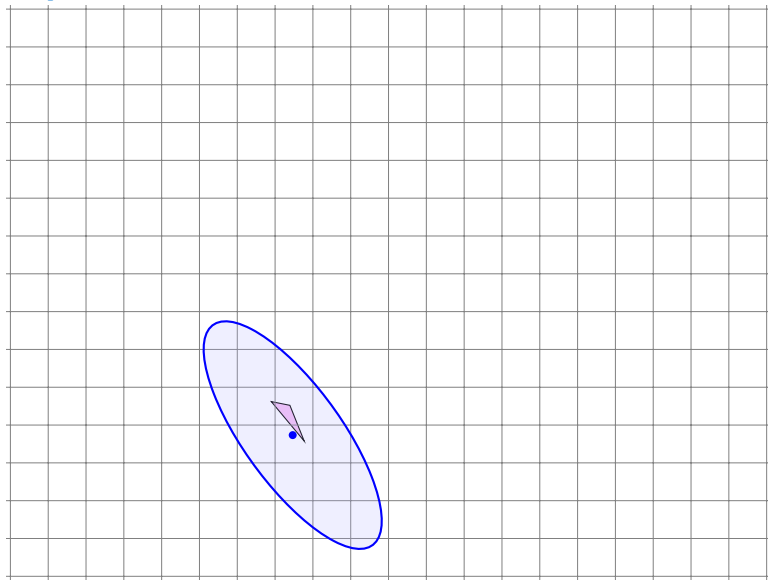
Example



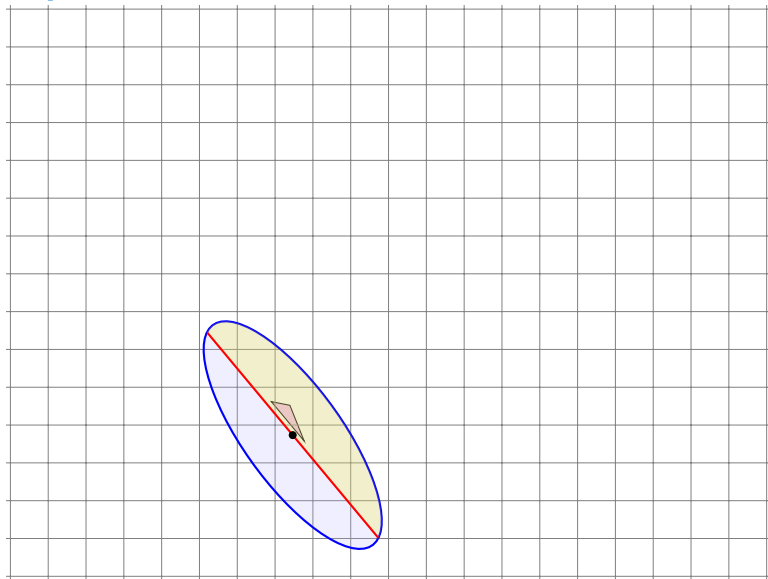
Example



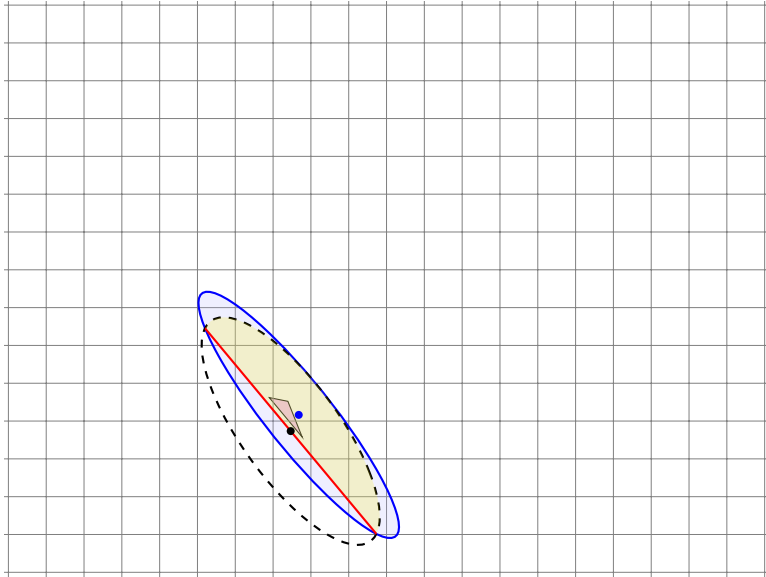
Example



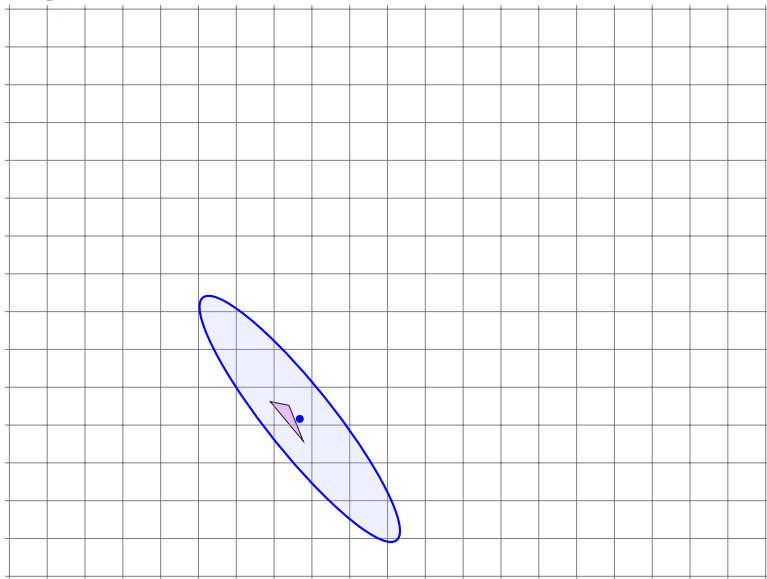
Example



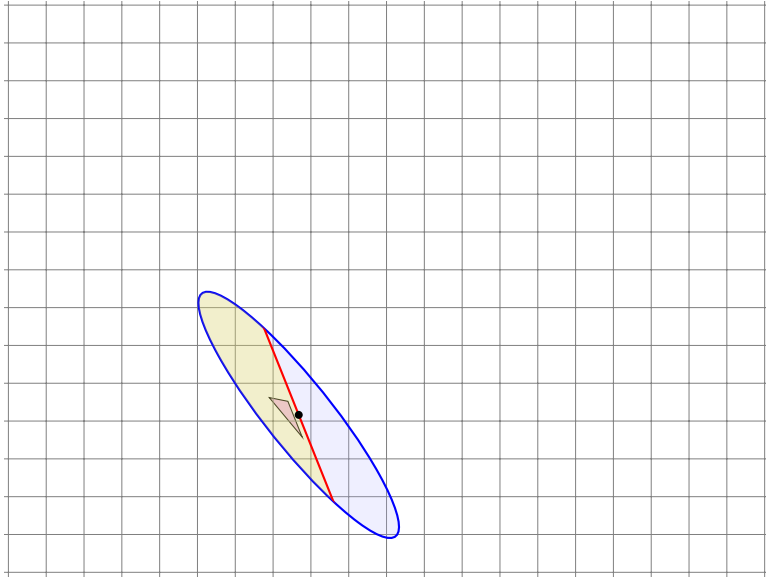
Example



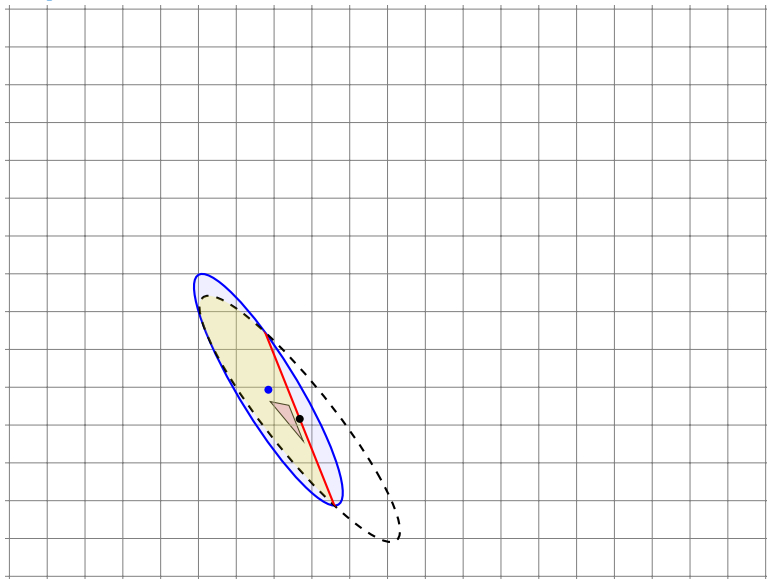
Example



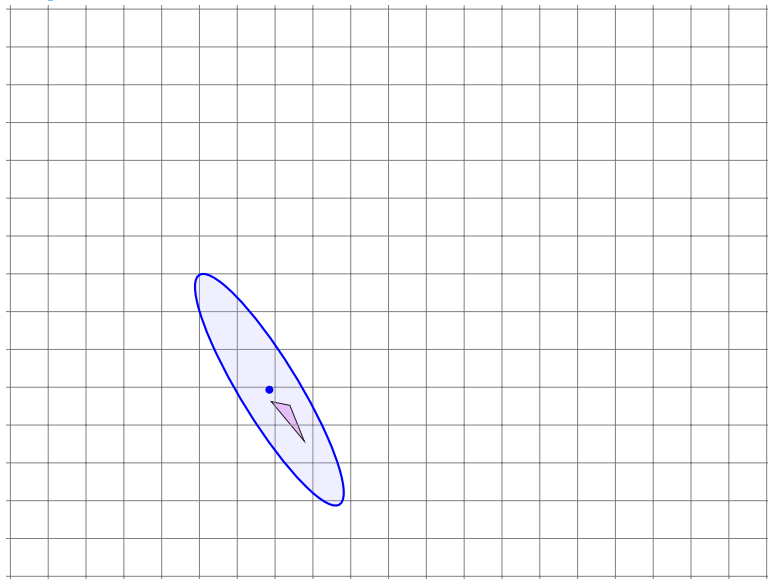
Example



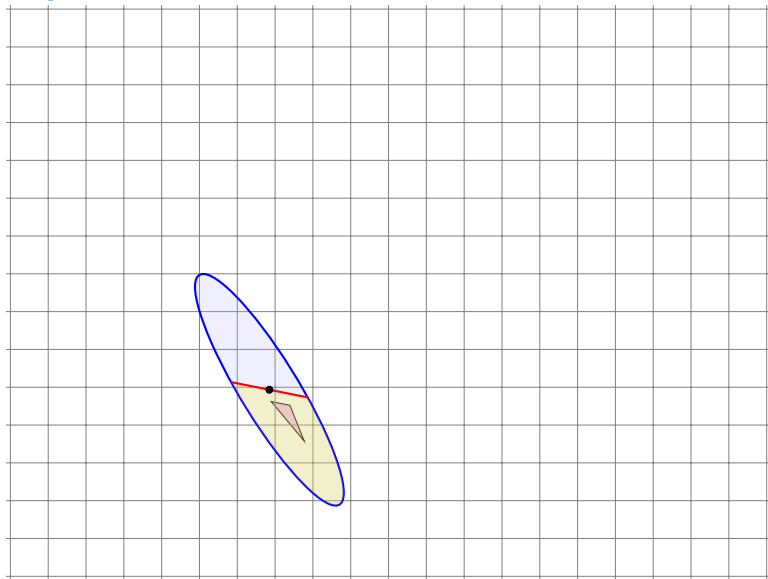
Example



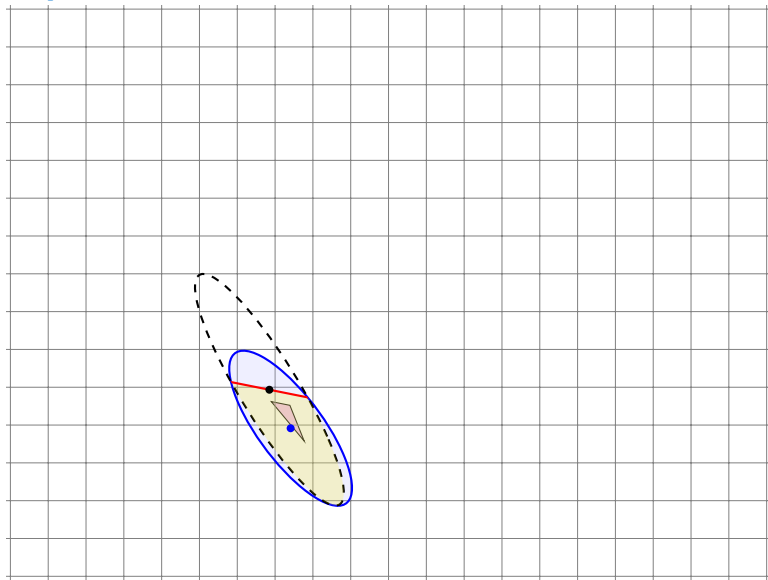
Example



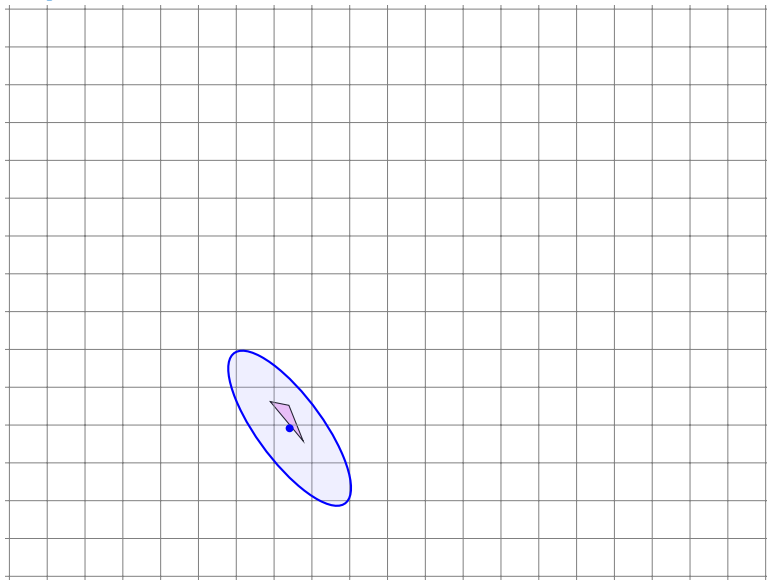
Example



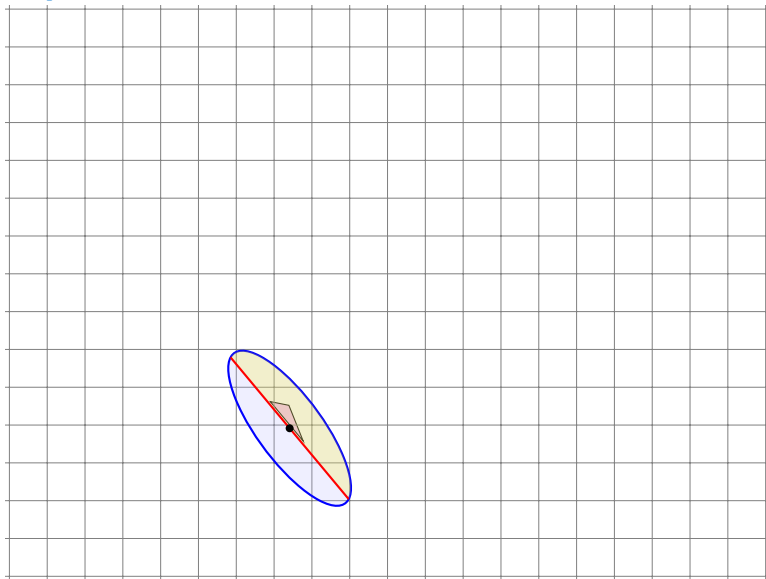
Example



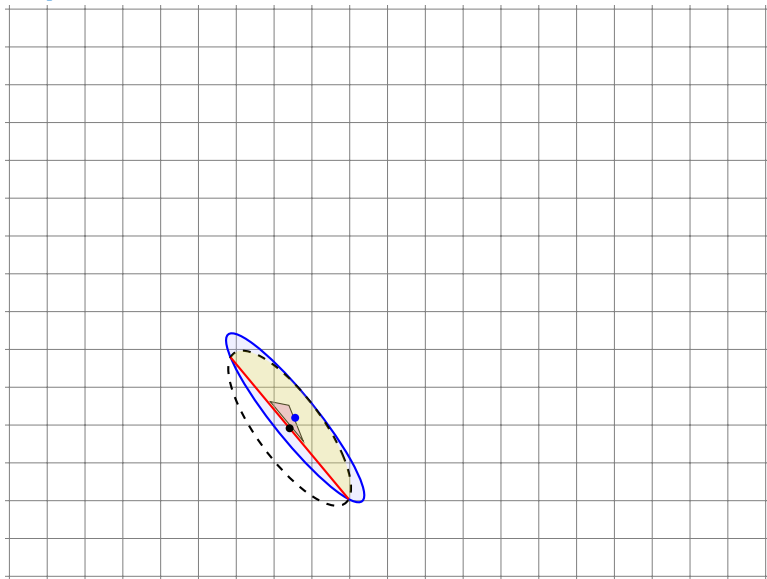
Example



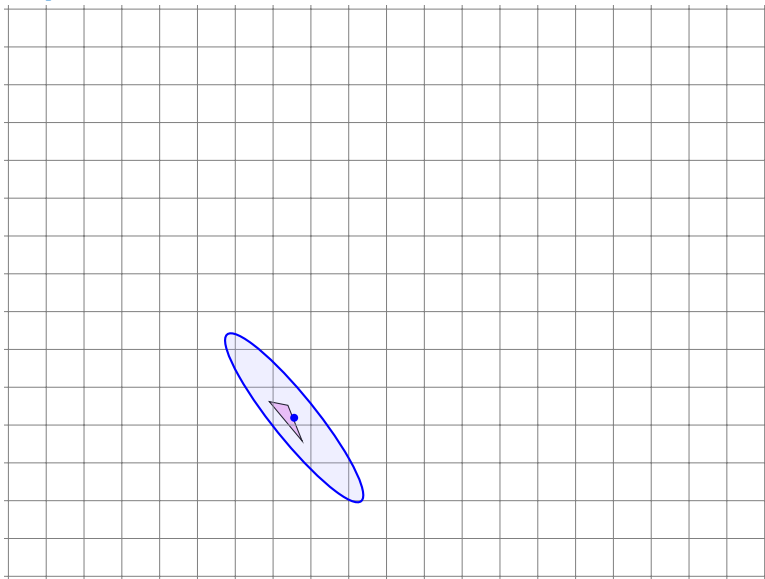
Example



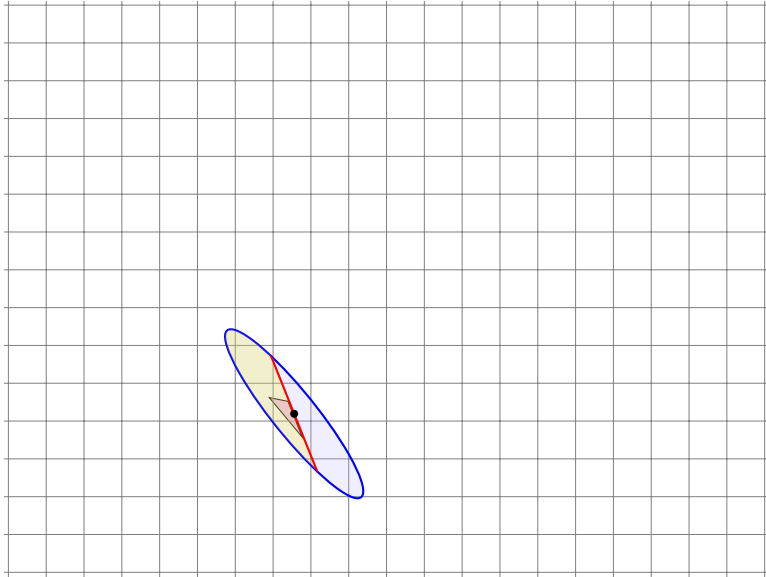
Example



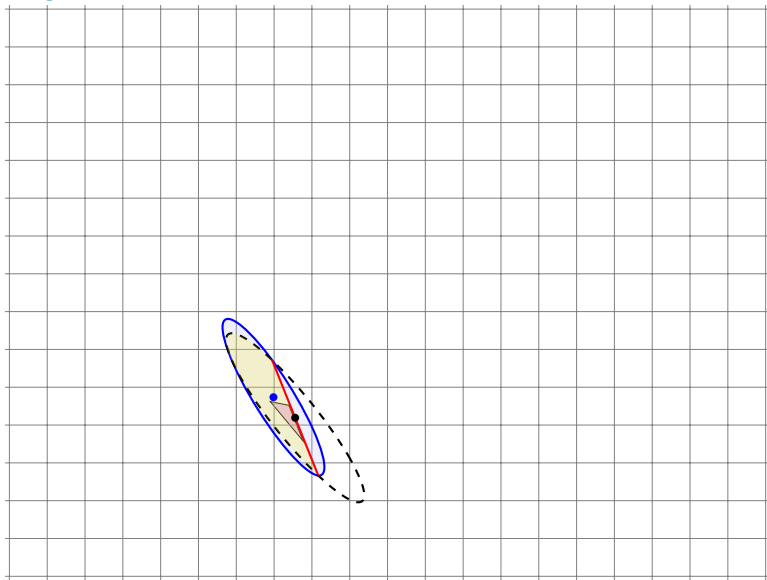
Example



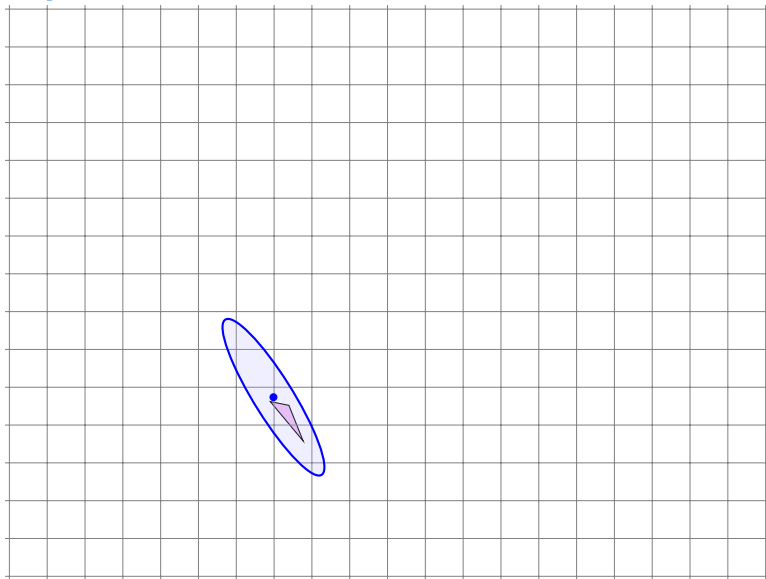
Example



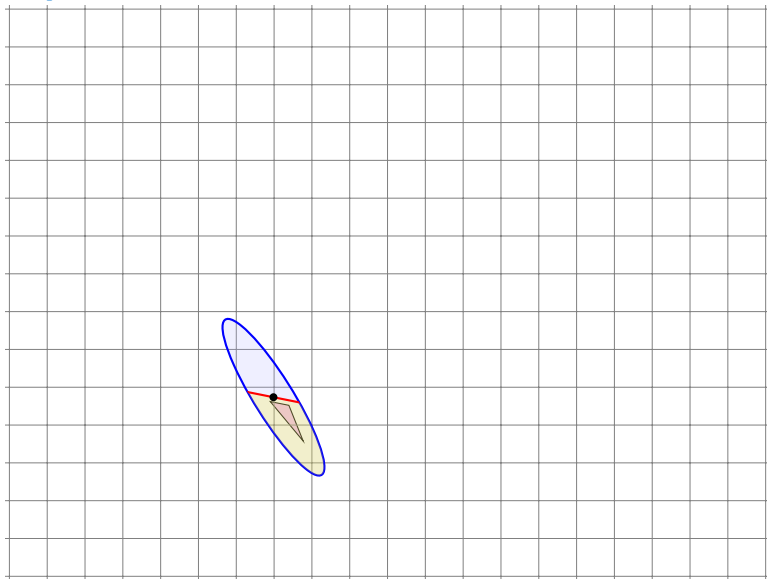
Example



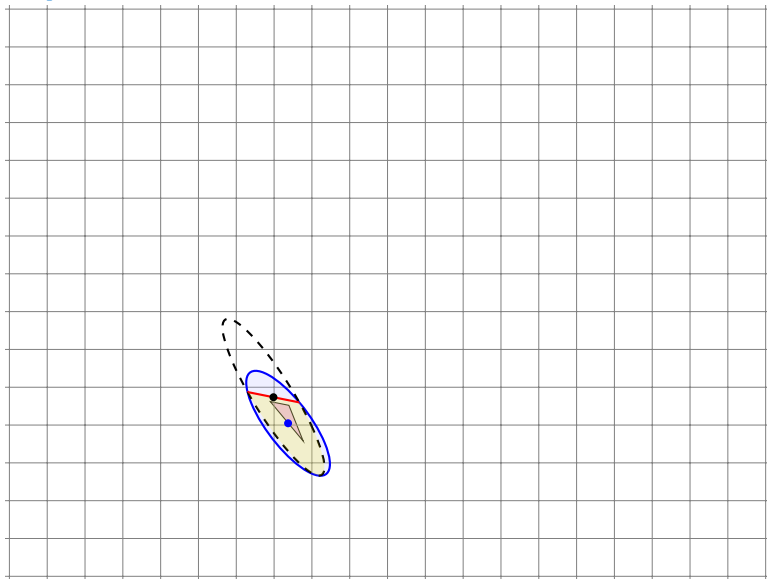
Example



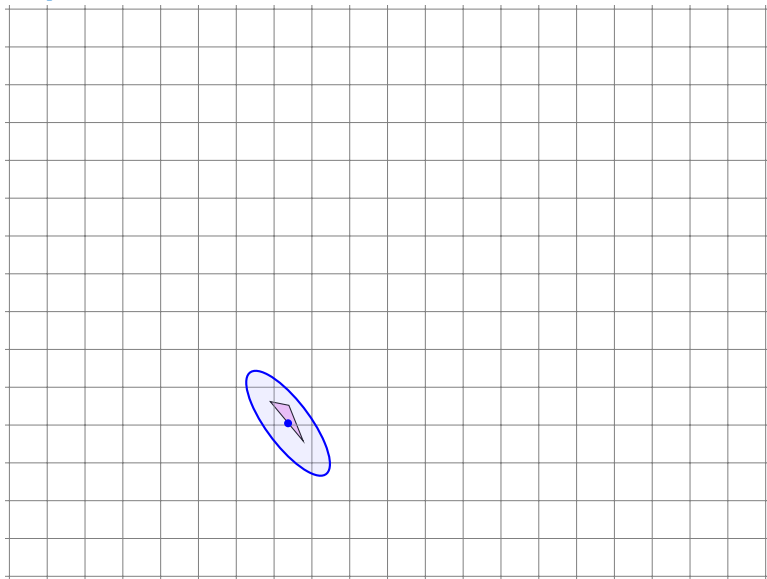
Example



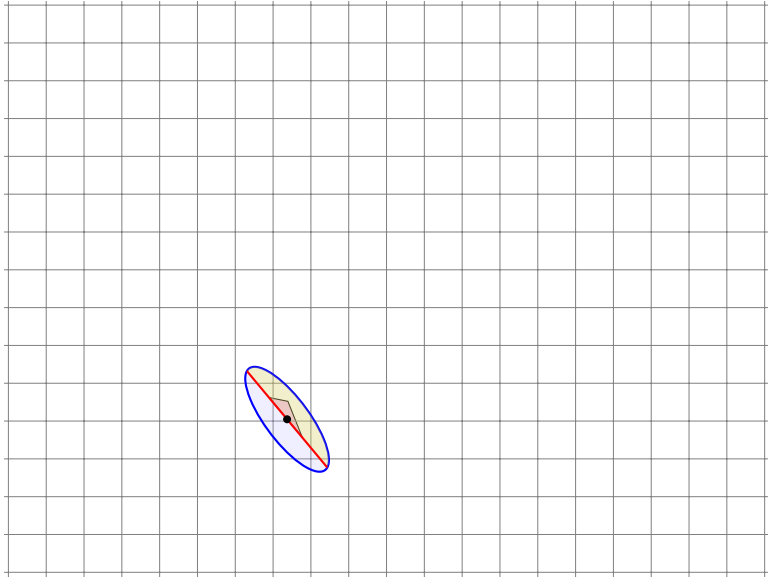
Example



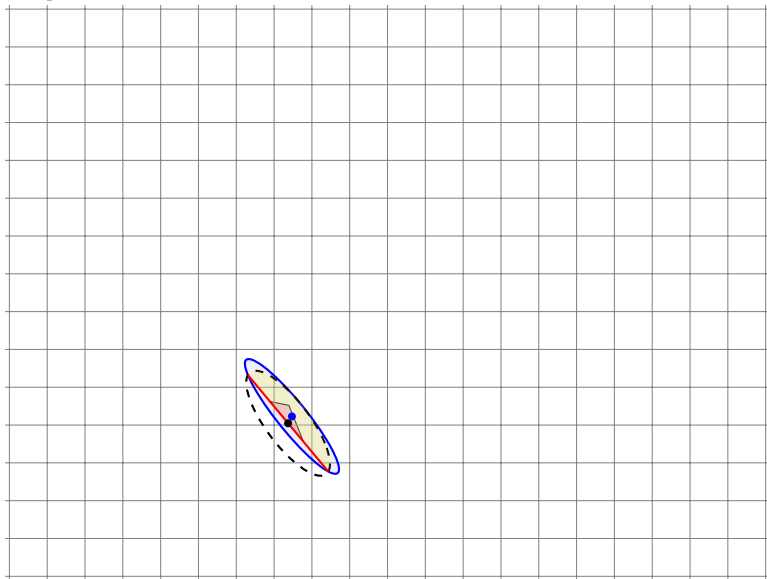
Example



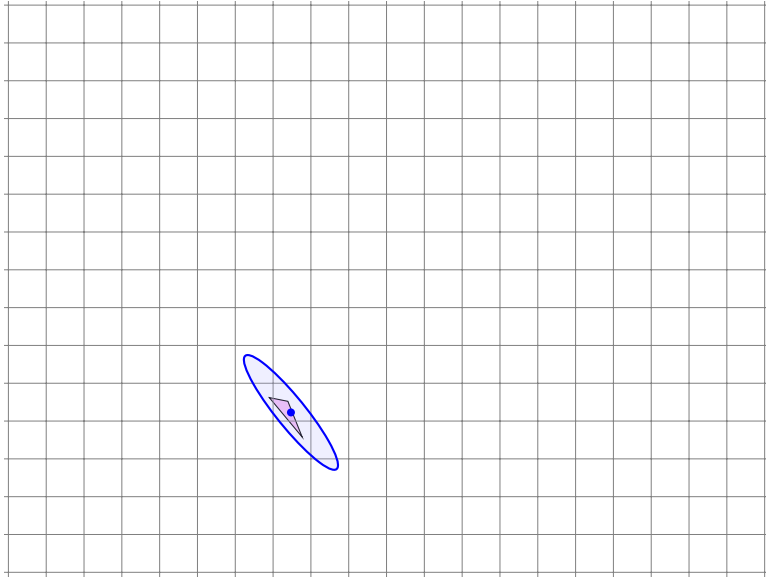
Example



Example



Example



Part III

Approximation Algorithms

There are many practically important optimization problems that are NP-hard.

What can we do?

- Heuristics
- Exploit special structure of instances occurring in practice
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimal

There are many practically important optimization problems that are NP-hard.

What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- ▶ **Heuristics.**
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

Definition 55

An α -approximation for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.

Why approximation algorithms?

We need algorithms for hard problems.

It gives a rigorous mathematical base for studying heuristics.

It provides a metric to compare the difficulty of various optimization problems.

Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Definition 56

An optimization problem $P = (\mathcal{I}, \text{sol}, m, \text{goal})$ is in **NPO** if

- ▶ $x \in \mathcal{I}$ can be **decided** in polynomial time
- ▶ $y \in \text{sol}(\mathcal{I})$ can be **verified** in polynomial time
- ▶ m can be computed in polynomial time
- ▶ $\text{goal} \in \{\min, \max\}$

In other words: the decision problem **is there a solution y with $m(x, y)$ at most/at least z** is in NP.

- ▶ x is problem instance
- ▶ y is candidate solution
- ▶ $m^*(x)$ cost/profit of an optimal solution

Definition 57 (Performance Ratio)

$$R(x, y) := \max \left\{ \frac{m(x, y)}{m^*(x)}, \frac{m^*(x)}{m(x, y)} \right\}$$

Definition 58 (r -approximation)

An algorithm A is an r -approximation algorithm iff

$$\forall x \in \mathcal{I} : R(x, A(x)) \leq r ,$$

and A runs in polynomial time.

Definition 59 (PTAS)

A PTAS for a problem P from NPO is an algorithm that takes as input $x \in \mathcal{I}$ and $\epsilon > 0$ and produces a solution y for x with

$$R(x, y) \leq 1 + \epsilon .$$

The running time is polynomial in $|x|$.

approximation with arbitrary good factor... fast?

Problems that have a PTAS

Scheduling. Given m jobs with known processing times; schedule the jobs on n machines such that the MAKESPAN is minimized.

Definition 60 (FPTAS)

An FPTAS for a problem P from NPO is an algorithm that takes as input $x \in \mathcal{I}$ and $\epsilon > 0$ and produces a solution y for x with

$$R(x, y) \leq 1 + \epsilon .$$

The running time is polynomial in $|x|$ and $1/\epsilon$.

approximation with arbitrary good factor... fast!

Problems that have an FPTAS

KNAPSACK. Given a set of items with profits and weights choose a subset of total weight at most W s.t. the profit is maximized.

Definition 61 (APX – approximable)

A problem P from NPO is in APX if there exist a constant $r \geq 1$ and an r -approximation algorithm for P .

constant factor approximation...

Problems that are in APX

MAXCUT. Given a graph $G = (V, E)$; partition V into two disjoint pieces A and B s. t. the number of edges between both pieces is maximized.

MAX-3SAT. Given a 3CNF-formula. Find an assignment to the variables that satisfies the maximum number of clauses.

Problems with polylogarithmic approximation guarantees

- ▶ Set Cover
- ▶ Minimum Multicut
- ▶ Sparsest Cut
- ▶ Minimum Bisection

There is an r -approximation with $r \leq \mathcal{O}(\log^c(|x|))$ for some constant c .

Note that only for some of the above problem a matching lower bound is known.

There are really difficult problems!

Theorem 62

For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$ -approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $P = NP$.

Note that an n -approximation is trivial.

There are really difficult problems!

Theorem 62

For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$ -approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $P = NP$.

Note that an n -approximation is trivial.

There are really difficult problems!

Theorem 62

For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$ -approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $P = NP$.

Note that an n -approximation is trivial.

There are weird problems!

Asymmetric k -Center admits an $\mathcal{O}(\log^* n)$ -approximation.

There is no $o(\log^* n)$ -approximation to Asymmetric k -Center unless $NP \subseteq DTIME(n^{\log \log \log n})$.

Class APX not important in practise.

Instead of saying **problem P is in APX** one says **problem P admits a 4-approximation.**

One only says that a problem is **APX-hard**.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore **Linear Programs** or **Integer Linear Programs** play a vital role in the design of many approximation algorithms.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore **Linear Programs** or **Integer Linear Programs** play a vital role in the design of many approximation algorithms.

Definition 63

An **Integer Linear Program** or **Integer Program** is a Linear Program in which all variables are required to be integral.

Definition 64

A **Mixed Integer Program** is a Linear Program in which a subset of the variables are required to be integral.

Definition 63

An **Integer Linear Program** or **Integer Program** is a Linear Program in which all variables are required to be integral.

Definition 64

A **Mixed Integer Program** is a Linear Program in which a subset of the variables are required to be integral.

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

Note that solving Integer Programs in general is NP-complete!

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

Note that solving Integer Programs in general is NP-complete!

Set Cover

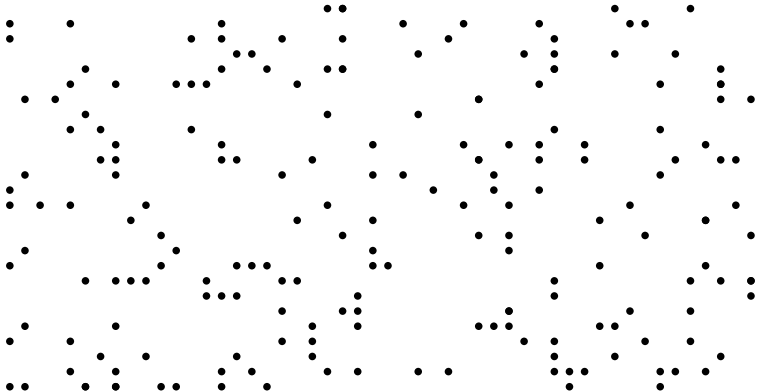
Given a ground set U , a collection of subsets $S_1, \dots, S_k \subseteq U$, where the i -th subset S_i has weight/cost w_i . Find a collection $I \subseteq \{1, \dots, k\}$ such that

$$\forall u \in U \exists i \in I: u \in S_i \text{ (every element is covered)}$$

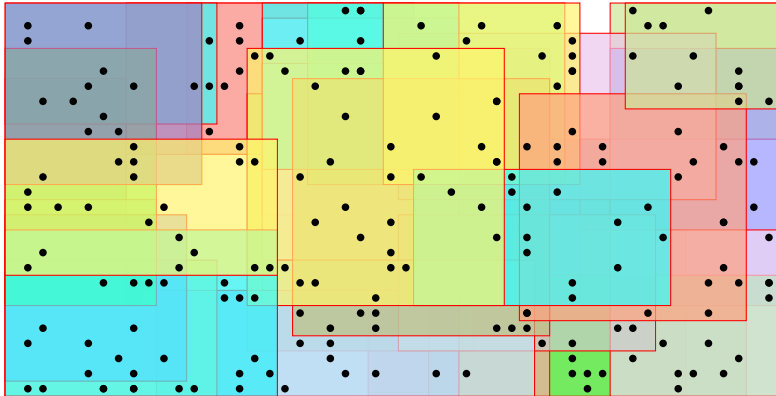
and

$$\sum_{i \in I} w_i \text{ is minimized.}$$

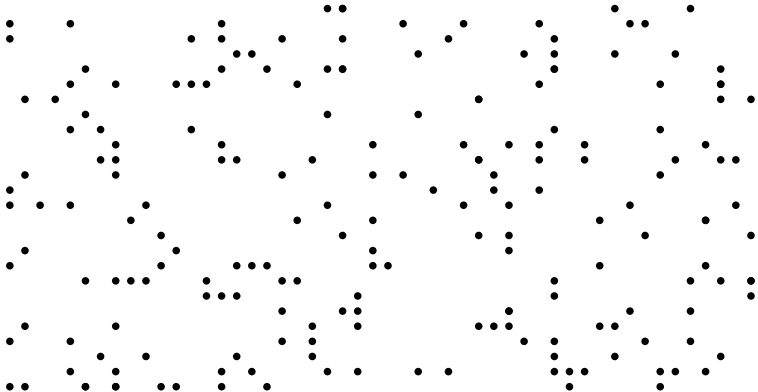
Set Cover



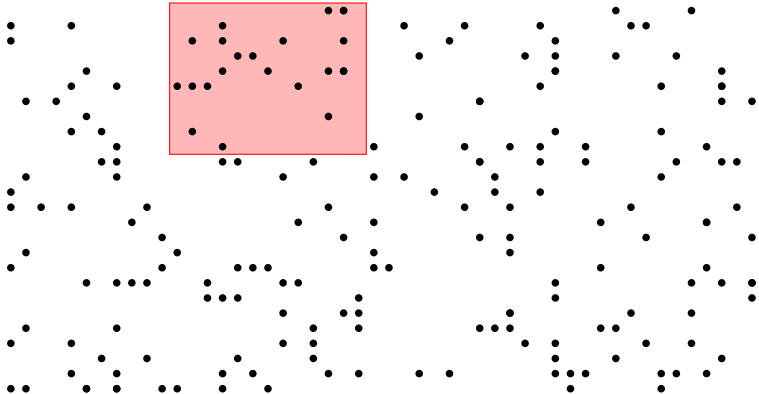
Set Cover



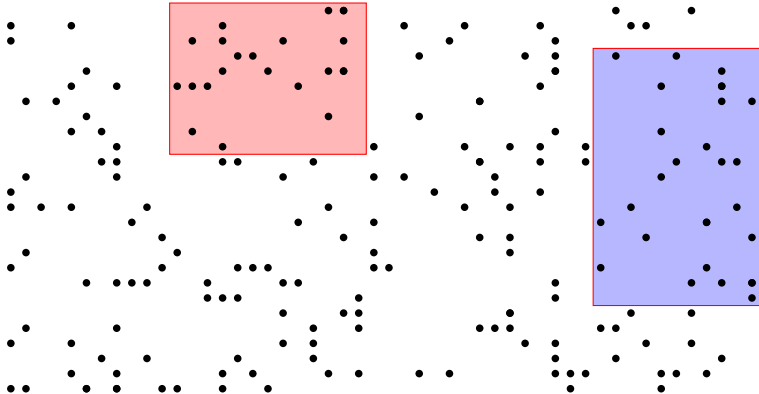
Set Cover



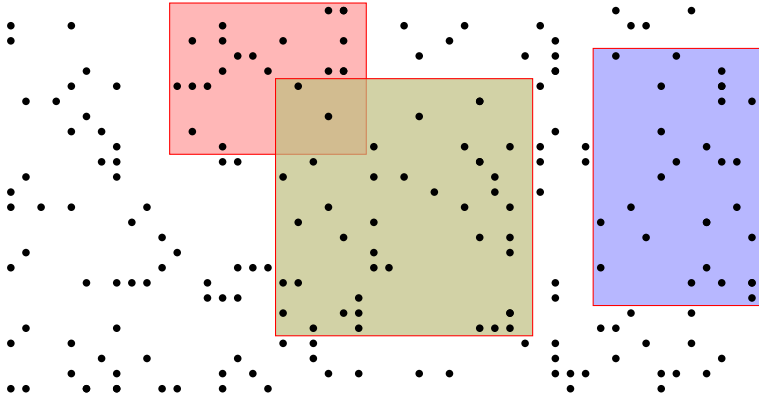
Set Cover



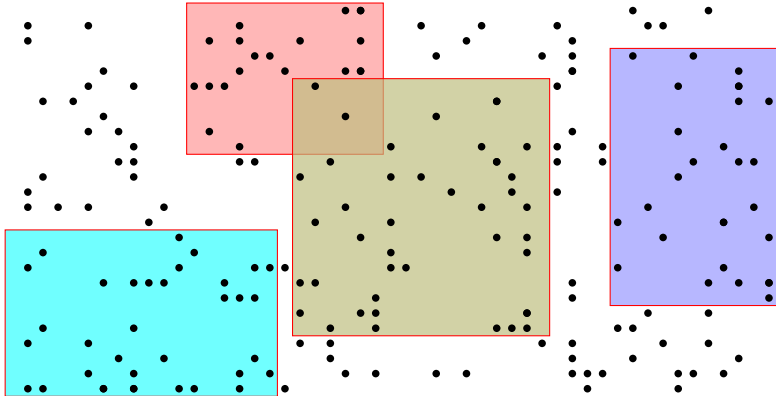
Set Cover



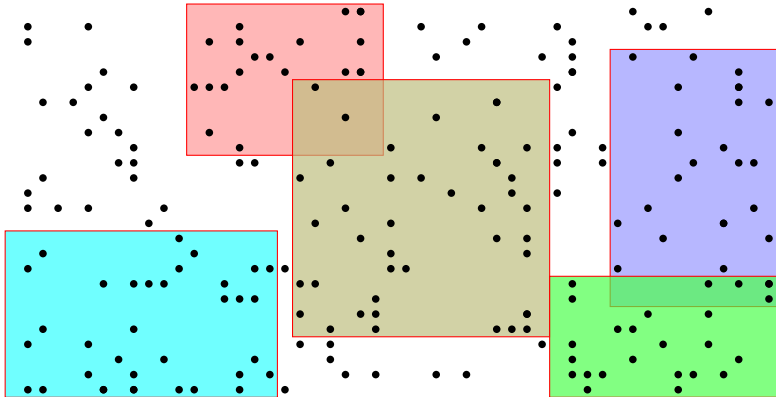
Set Cover



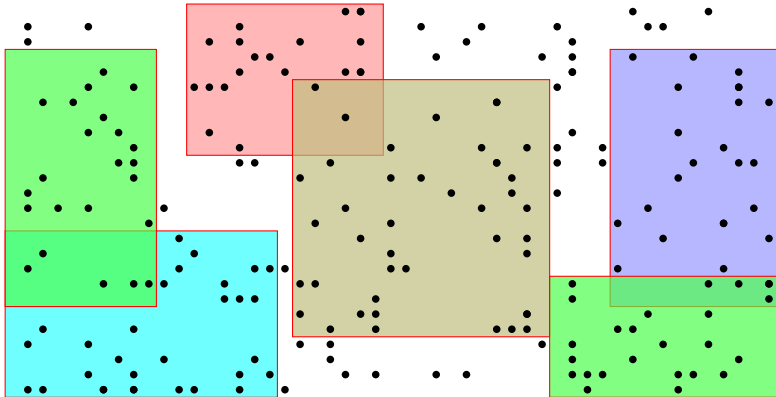
Set Cover



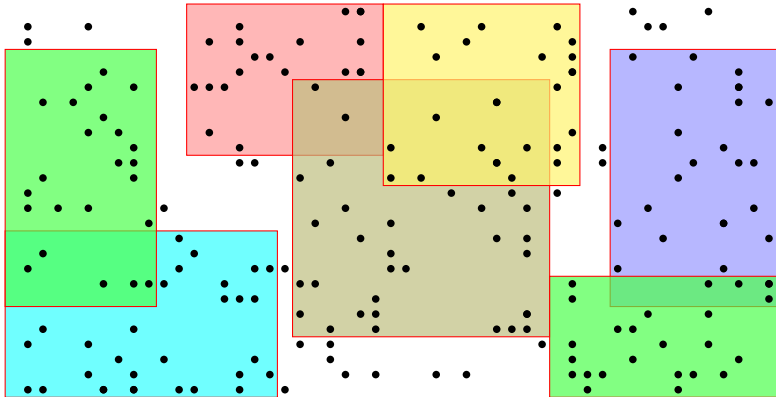
Set Cover



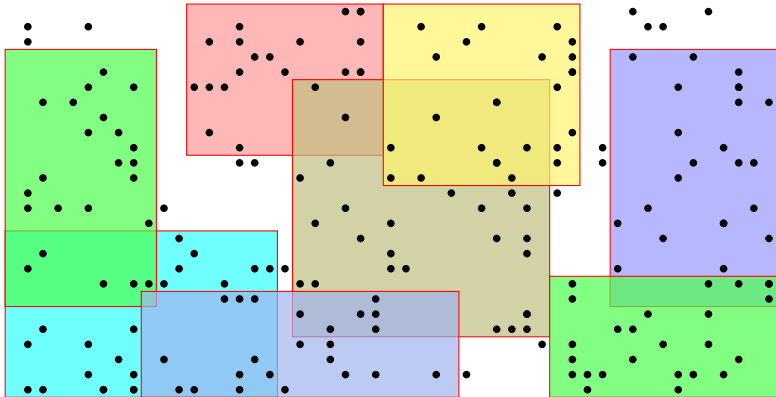
Set Cover



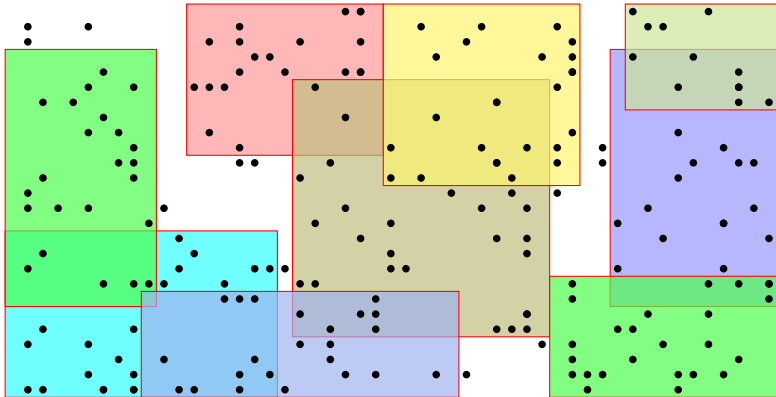
Set Cover



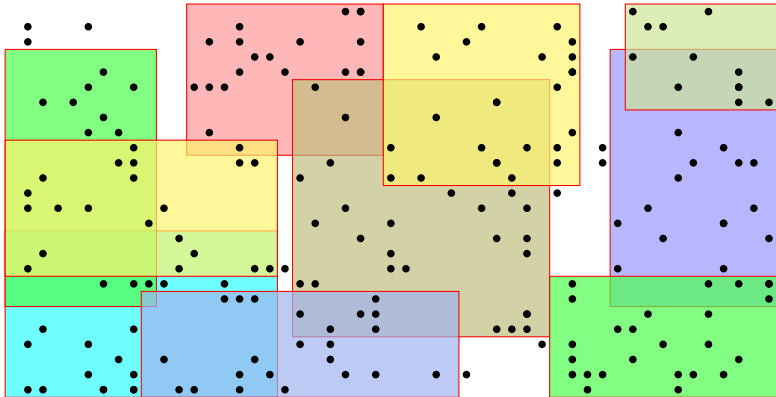
Set Cover



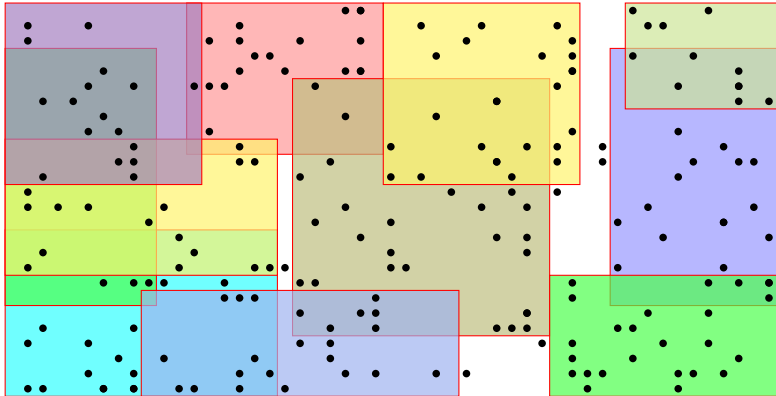
Set Cover



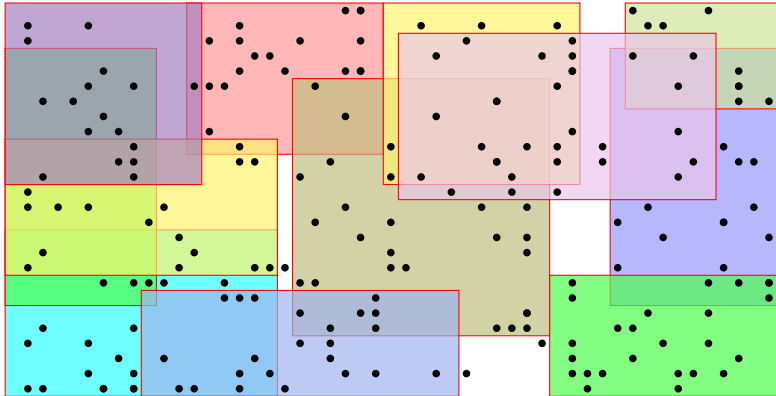
Set Cover



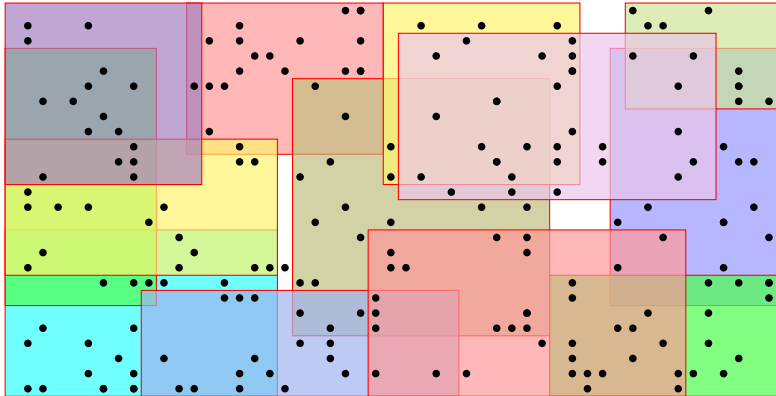
Set Cover



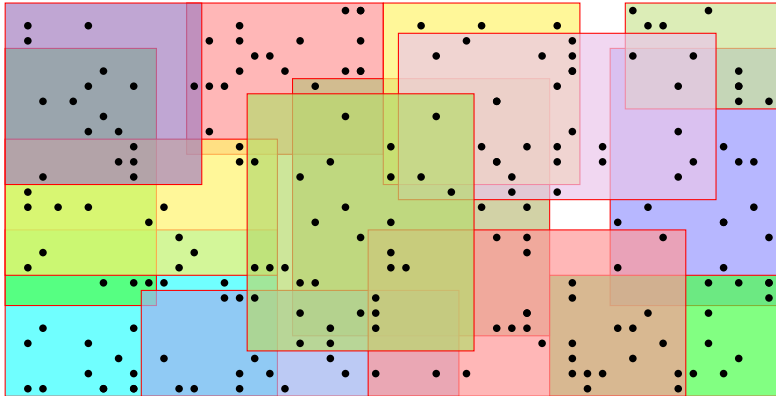
Set Cover



Set Cover



Set Cover



IP-Formulation of Set Cover

$$\begin{array}{llllll} \min & & \sum_i w_i x_i & & & \\ \text{s.t.} & \forall u \in U & \sum_{i: u \in S_i} x_i & \geq & 1 & \\ & \forall i \in \{1, \dots, k\} & x_i & \geq & 0 & \\ & \forall i \in \{1, \dots, k\} & x_i & \text{integral} & & \end{array}$$

Vertex Cover

Given a graph $G = (V, E)$ and a weight w_v for every node. Find a vertex subset $S \subseteq V$ of minimum weight such that every edge is incident to at least one vertex in S .

IP-Formulation of Vertex Cover

$$\begin{array}{ll}\min & \sum_{v \in V} w_v x_v \\ \text{s.t.} & \forall e = (i, j) \in E \quad x_i + x_j \geq 1 \\ & \forall v \in V \quad x_v \in \{0, 1\}\end{array}$$

Maximum Weighted Matching

Given a graph $G = (V, E)$, and a weight w_e for every edge $e \in E$. Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

$$\begin{array}{ll} \max & \sum_{e \in E} w_e x_e \\ \text{s.t.} & \forall v \in V \quad \sum_{e: v \in e} x_e \leq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Maximum Weighted Matching

Given a graph $G = (V, E)$, and a weight w_e for every edge $e \in E$. Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

$$\begin{array}{ll} \max & \sum_{e \in E} w_e x_e \\ \text{s.t.} & \forall v \in V \quad \sum_{e: v \in e} x_e \leq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Maximum Independent Set

Given a graph $G = (V, E)$, and a weight w_v for every node $v \in V$. Find a subset $S \subseteq V$ of nodes of maximum weight such that no two vertices in S are adjacent.

$$\begin{array}{ll} \max & \sum_{v \in V} w_v x_v \\ \text{s.t.} & \forall e = (i, j) \in E \quad x_i + x_j \leq 1 \\ & \forall v \in V \quad x_v \in \{0, 1\} \end{array}$$

Maximum Independent Set

Given a graph $G = (V, E)$, and a weight w_v for every node $v \in V$. Find a subset $S \subseteq V$ of nodes of maximum weight such that no two vertices in S are adjacent.

$$\begin{array}{ll} \max & \sum_{v \in V} w_v x_v \\ \text{s.t.} & \forall e = (i, j) \in E \quad x_i + x_j \leq 1 \\ & \forall v \in V \quad x_v \in \{0, 1\} \end{array}$$

Knapsack

Given a set of items $\{1, \dots, n\}$, where the i -th item has weight w_i and profit p_i , and given a threshold K . Find a subset $I \subseteq \{1, \dots, n\}$ of items of total weight at most K such that the profit is maximized.

$$\begin{array}{ll} \max & \sum_{i=1}^n p_i x_i \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \leq K \\ & \forall i \in \{1, \dots, n\} \quad x_i \in \{0, 1\} \end{array}$$

Knapsack

Given a set of items $\{1, \dots, n\}$, where the i -th item has weight w_i and profit p_i , and given a threshold K . Find a subset $I \subseteq \{1, \dots, n\}$ of items of total weight at most K such that the profit is maximized.

$$\begin{array}{ll} \max & \sum_{i=1}^n p_i x_i \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \leq K \\ & \forall i \in \{1, \dots, n\} \quad x_i \in \{0, 1\} \end{array}$$

Definition 65

A linear program LP is a **relaxation** of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

We obtain a relaxation for all examples by writing $x_i \in [0, 1]$ instead of $x_i \in \{0, 1\}$.

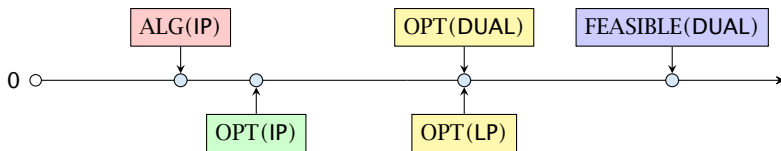
Definition 65

A linear program LP is a **relaxation** of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

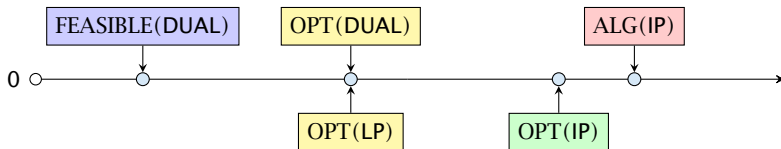
We obtain a relaxation for all examples by writing $x_i \in [0, 1]$ instead of $x_i \in \{0, 1\}$.

By solving a relaxation we obtain an upper bound for a maximization problem and a lower bound for a minimization problem.

Maximization Problems:



Minimization Problems:



Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

Set Cover relaxation:

$$\begin{array}{ll} \min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i: u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \in [0, 1] \end{array}$$

Let f_u be the number of sets that the element u is contained in (the frequency of u). Let $f = \max_u \{f_u\}$ be the maximum frequency.

Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

Set Cover relaxation:

$$\begin{array}{ll} \min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i: u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \in [0, 1] \end{array}$$

Let f_u be the number of sets that the element u is contained in (the frequency of u). Let $f = \max_u \{f_u\}$ be the maximum frequency.

Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

Set Cover relaxation:

$$\begin{array}{ll} \min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i: u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \in [0, 1] \end{array}$$

Let f_u be the number of sets that the element u is contained in (the frequency of u). Let $f = \max_u \{f_u\}$ be the maximum frequency.

Technique 1: Round the LP solution.

Rounding Algorithm:

Set all x_i -values with $x_i \geq \frac{1}{f}$ to 1. Set all other x_i -values to 0.

Technique 1: Round the LP solution.

Lemma 66

The rounding algorithm gives an f -approximation.

Proof: Every $u \in U$ is covered.

By Lemma 65, we know that $\sum_{S \in \mathcal{S}} x_S \cdot |S| \geq f \cdot |U|$.
The sum contains at most f elements.
Therefore, one of the sets that remain is must have size at least $|U|$.
This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 66

The rounding algorithm gives an f -approximation.

Proof: Every $u \in U$ is covered.

Technique 1: Round the LP solution.

Lemma 66

The rounding algorithm gives an f -approximation.

Proof: Every $u \in U$ is covered.

- ▶ We know that $\sum_{i:u \in S_i} x_i \geq 1$.
- ▶ The sum contains at most $f_u \leq f$ elements.
- ▶ Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- ▶ This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 66

The rounding algorithm gives an f -approximation.

Proof: Every $u \in U$ is covered.

- ▶ We know that $\sum_{i: u \in S_i} x_i \geq 1$.
- ▶ The sum contains at most $f_u \leq f$ elements.
- ▶ Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- ▶ This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 66

The rounding algorithm gives an f -approximation.

Proof: Every $u \in U$ is covered.

- ▶ We know that $\sum_{i:u \in S_i} x_i \geq 1$.
- ▶ The sum contains at most $f_u \leq f$ elements.
- ▶ Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- ▶ This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 66

The rounding algorithm gives an f -approximation.

Proof: Every $u \in U$ is covered.

- ▶ We know that $\sum_{i:u \in S_i} x_i \geq 1$.
- ▶ The sum contains at most $f_u \leq f$ elements.
- ▶ Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- ▶ This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

$$\sum_{i \in I} w_i$$

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

$$\sum_{i \in I} w_i \leq \sum_{i=1}^k w_i (f \cdot x_i)$$

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

$$\begin{aligned}\sum_{i \in I} w_i &\leq \sum_{i=1}^k w_i (f \cdot x_i) \\ &= f \cdot \text{cost}(x)\end{aligned}$$

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

$$\begin{aligned}\sum_{i \in I} w_i &\leq \sum_{i=1}^k w_i (f \cdot x_i) \\ &= f \cdot \text{cost}(x) \\ &\leq f \cdot \text{OPT} .\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Relaxation for Set Cover

Primal:

$$\begin{array}{ll}\min & \sum_{i \in I} w_i x_i \\ \text{s.t. } \forall u & \sum_{i: u \in S_i} x_i \geq 1 \\ & x_i \geq 0\end{array}$$

Dual:

$$\begin{array}{ll}\max & \sum_{u \in U} y_u \\ \text{s.t. } \forall i & \sum_{u: u \in S_i} y_u \leq w_i \\ & y_u \geq 0\end{array}$$

Technique 2: Rounding the Dual Solution.

Relaxation for Set Cover

Primal:

$$\begin{array}{ll}\min & \sum_{i \in I} w_i x_i \\ \text{s.t. } \forall u & \sum_{i: u \in S_i} x_i \geq 1 \\ & x_i \geq 0\end{array}$$

Dual:

$$\begin{array}{ll}\max & \sum_{u \in U} y_u \\ \text{s.t. } \forall i & \sum_{u: u \in S_i} y_u \leq w_i \\ & y_u \geq 0\end{array}$$

Technique 2: Rounding the Dual Solution.

Relaxation for Set Cover

Primal:

$$\begin{array}{ll}\min & \sum_{i \in I} w_i x_i \\ \text{s.t. } \forall u & \sum_{i: u \in S_i} x_i \geq 1 \\ & x_i \geq 0\end{array}$$

Dual:

$$\begin{array}{ll}\max & \sum_{u \in U} y_u \\ \text{s.t. } \forall i & \sum_{u: u \in S_i} y_u \leq w_i \\ & y_u \geq 0\end{array}$$

Technique 2: Rounding the Dual Solution.

Rounding Algorithm:

Let I denote the index set of sets for which the dual constraint is tight. This means for all $i \in I$

$$\sum_{u:u \in S_i} y_u = w_i$$

Technique 2: Rounding the Dual Solution.

Lemma 67

The resulting index set is an f -approximation.

Proof:

Every $u \in U$ is covered.

Suppose there is a u that is not covered.

This means $\sum_{i \in I} x_i \cdot \text{cover}(u, i) < 1$. But all x_i are 1 that remains

but then we could be increased the dual solution without violating any constraints. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Lemma 67

The resulting index set is an f -approximation.

Proof:

Every $u \in U$ is covered.

Technique 2: Rounding the Dual Solution.

Lemma 67

The resulting index set is an f -approximation.

Proof:

Every $u \in U$ is covered.

- ▶ Suppose there is a u that is not covered.
- ▶ This means $\sum_{u: u \in S_i} y_u < w_i$ for all sets S_i that contain u .
- ▶ But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Lemma 67

The resulting index set is an f -approximation.

Proof:

Every $u \in U$ is covered.

- ▶ Suppose there is a u that is not covered.
- ▶ This means $\sum_{u: u \in S_i} y_u < w_i$ for all sets S_i that contain u .
- ▶ But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Lemma 67

The resulting index set is an f -approximation.

Proof:

Every $u \in U$ is covered.

- ▶ Suppose there is a u that is not covered.
- ▶ This means $\sum_{u: u \in S_i} y_u < w_i$ for all sets S_i that contain u .
- ▶ But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Proof:

$$\sum_{i \in I} w_i$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\sum_{i \in I} w_i = \sum_{i \in I} \sum_{u: u \in S_i} y_u$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i \in I} w_i &= \sum_{i \in I} \sum_{u: u \in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i \in I} w_i &= \sum_{i \in I} \sum_{u: u \in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u \\ &\leq \sum_u f_u y_u\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i \in I} w_i &= \sum_{i \in I} \sum_{u: u \in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u \\ &\leq \sum_u f_u y_u \\ &\leq f \sum_u y_u\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i \in I} w_i &= \sum_{i \in I} \sum_{u: u \in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u \\ &\leq \sum_u f_u y_u \\ &\leq f \sum_u y_u \\ &\leq f \text{cost}(x^*)\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i \in I} w_i &= \sum_{i \in I} \sum_{u: u \in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u \\ &\leq \sum_u f_u y_u \\ &\leq f \sum_u y_u \\ &\leq f \text{cost}(x^*) \\ &\leq f \cdot \text{OPT}\end{aligned}$$

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means I' is never better than I .

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means I' is never better than I .

- ▶ Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- ▶ This means $x_i \geq \frac{1}{f}$.
- ▶ Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- ▶ Hence, the second algorithm will also choose S_i .

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means I' is never better than I .

- ▶ Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- ▶ This means $x_i \geq \frac{1}{f}$.
 - ▶ Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
 - ▶ Hence, the second algorithm will also choose S_i .

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means I' is never better than I .

- ▶ Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- ▶ This means $x_i \geq \frac{1}{f}$.
- ▶ Because of **Complementary Slackness Conditions** the corresponding constraint in the dual must be tight.
- ▶ Hence, the second algorithm will also choose S_i .

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means I' is never better than I .

- ▶ Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- ▶ This means $x_i \geq \frac{1}{f}$.
- ▶ Because of **Complementary Slackness Conditions** the corresponding constraint in the dual must be tight.
- ▶ Hence, the second algorithm will also choose S_i .

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f -approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

The solution is dual feasible and I is a cover.

The cost of the solution is at most f times the cost of the LP.

The cost of the solution is at least the cost of the LP.

Therefore, the solution is an f -approximation to the primal LP.

The only remaining task is to find a dual feasible solution I such that I is a cover.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f -approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

Let I be a set of edges. We call I a *feasible solution* if $|I \cap e| \leq 1$ for every edge e of G . We call I a *cover* if $|I \cap e| \geq 1$ for every edge e of G .

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f -approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f -approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f -approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

Algorithm 1 PrimalDual

```
1:  $\gamma \leftarrow 0$ 
2:  $I \leftarrow \emptyset$ 
3: while exists  $u \notin \bigcup_{i \in I} S_i$  do
4:   increase dual variable  $\gamma_u$  until constraint for some
     new set  $S_\ell$  becomes tight
5:    $I \leftarrow I \cup \{\ell\}$ 
```

Technique 4: The Greedy Algorithm

Algorithm 1 Greedy

```
1:  $I \leftarrow \emptyset$ 
2:  $\hat{S}_j \leftarrow S_j$    for all  $j$ 
3: while  $I$  not a set cover do
4:    $\ell \leftarrow \arg \min_{j: \hat{S}_j \neq \emptyset} \frac{w_j}{|\hat{S}_j|}$ 
5:    $I \leftarrow I \cup \{\ell\}$ 
6:    $\hat{S}_j \leftarrow \hat{S}_j - S_\ell$    for all  $j$ 
```

In every round the Greedy algorithm takes the set that covers remaining elements in the most **cost-effective** way.

We choose a set such that the ratio between cost and still uncovered elements in the set is minimized.

Technique 4: The Greedy Algorithm

Lemma 68

Given positive numbers $a_1, \dots, a_k$ and $b_1, \dots, b_k$, and $S \subseteq \{1, \dots, k\}$ then

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_{i \in S} a_i}{\sum_{i \in S} b_i} \leq \max_i \frac{a_i}{b_i}$$

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
$$w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}.$$

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ . $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT .

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence,
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Adding this set to our solution means $n_{\ell+1} = n_{\ell} - |\hat{S}_j|$.

$$w_j \leq \frac{|\hat{S}_j| \cdot \text{OPT}}{n_{\ell}} = \frac{n_{\ell} - n_{\ell+1}}{n_{\ell}} \cdot \text{OPT}$$

Technique 4: The Greedy Algorithm

Adding this set to our solution means $n_{\ell+1} = n_{\ell} - |\hat{S}_j|$.

$$w_j \leq \frac{|\hat{S}_j| \text{OPT}}{n_{\ell}} = \frac{n_{\ell} - n_{\ell+1}}{n_{\ell}} \cdot \text{OPT}$$

Technique 4: The Greedy Algorithm

$$\sum_{j \in I} w_j$$

Technique 4: The Greedy Algorithm

$$\sum_{j \in I} w_j \leq \sum_{\ell=1}^s \frac{n_{\ell} - n_{\ell+1}}{n_{\ell}} \cdot \text{OPT}$$

Technique 4: The Greedy Algorithm

$$\begin{aligned}\sum_{j \in I} w_j &\leq \sum_{\ell=1}^s \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \text{OPT} \\ &\leq \text{OPT} \sum_{\ell=1}^s \left(\frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \cdots + \frac{1}{n_{\ell+1} + 1} \right)\end{aligned}$$

Technique 4: The Greedy Algorithm

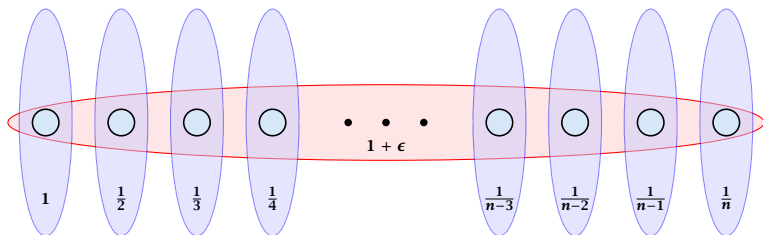
$$\begin{aligned}\sum_{j \in I} w_j &\leq \sum_{\ell=1}^s \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \text{OPT} \\ &\leq \text{OPT} \sum_{\ell=1}^s \left(\frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \cdots + \frac{1}{n_{\ell+1} + 1} \right) \\ &= \text{OPT} \sum_{i=1}^n \frac{1}{i}\end{aligned}$$

Technique 4: The Greedy Algorithm

$$\begin{aligned}\sum_{j \in I} w_j &\leq \sum_{\ell=1}^s \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \text{OPT} \\ &\leq \text{OPT} \sum_{\ell=1}^s \left(\frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \cdots + \frac{1}{n_{\ell+1} + 1} \right) \\ &= \text{OPT} \sum_{i=1}^n \frac{1}{i} \\ &= H_n \cdot \text{OPT} \leq \text{OPT}(\ln n + 1) \ .\end{aligned}$$

Technique 4: The Greedy Algorithm

A tight example:



Technique 5: Randomized Rounding

One round of randomized rounding:

Pick set S_j uniformly at random with probability $1 - x_j$ (for all j).

Version A: Repeat rounds until you nearly have a cover. Cover remaining elements by some simple heuristic.

Version B: Repeat for s rounds. If you have a cover STOP. Otherwise, repeat the whole algorithm.

Technique 5: Randomized Rounding

One round of randomized rounding:

Pick set S_j uniformly at random with probability $1 - x_j$ (for all j).

Version A: Repeat rounds until you nearly have a cover. Cover remaining elements by some simple heuristic.

Version B: Repeat for s rounds. If you have a cover STOP.
Otherwise, repeat the whole algorithm.

Technique 5: Randomized Rounding

One round of randomized rounding:

Pick set S_j uniformly at random with probability $1 - x_j$ (for all j).

Version A: Repeat rounds until you nearly have a cover. Cover remaining elements by some simple heuristic.

Version B: Repeat for s rounds. If you have a cover STOP. Otherwise, repeat the whole algorithm.

Probability that $u \in U$ is not covered (in one round):

$$\Pr[u \text{ not covered in one round}]$$

Probability that $u \in U$ is not covered (in one round):

$\Pr[u \text{ not covered in one round}]$

$$= \prod_{j: u \in S_j} (1 - x_j)$$

Probability that $u \in U$ is not covered (in one round):

$\Pr[u \text{ not covered in one round}]$

$$= \prod_{j: u \in S_j} (1 - x_j) \leq \prod_{j: u \in S_j} e^{-x_j}$$

Probability that $u \in U$ is not covered (in one round):

$\Pr[u \text{ not covered in one round}]$

$$\begin{aligned} &= \prod_{j: u \in S_j} (1 - x_j) \leq \prod_{j: u \in S_j} e^{-x_j} \\ &= e^{-\sum_{j: u \in S_j} x_j} \end{aligned}$$

Probability that $u \in U$ is not covered (in one round):

$\Pr[u \text{ not covered in one round}]$

$$\begin{aligned} &= \prod_{j: u \in S_j} (1 - x_j) \leq \prod_{j: u \in S_j} e^{-x_j} \\ &= e^{-\sum_{j: u \in S_j} x_j} \leq e^{-1} . \end{aligned}$$

Probability that $u \in U$ is not covered (in one round):

$$\begin{aligned}\Pr[u \text{ not covered in one round}] &= \prod_{j: u \in S_j} (1 - x_j) \leq \prod_{j: u \in S_j} e^{-x_j} \\ &= e^{-\sum_{j: u \in S_j} x_j} \leq e^{-1} .\end{aligned}$$

Probability that $u \in U$ is not covered (after ℓ rounds):

$$\Pr[u \text{ not covered after } \ell \text{ round}] \leq \frac{1}{e^\ell} .$$

$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$

$$\begin{aligned} & \Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\ &= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \dots \vee u_n \text{ not covered}] \end{aligned}$$

$$\begin{aligned} & \Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\ &= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \dots \vee u_n \text{ not covered}] \\ &\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \end{aligned}$$

$$\begin{aligned} & \Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\ &= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \dots \vee u_n \text{ not covered}] \\ &\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \leq ne^{-\ell} . \end{aligned}$$

$$\begin{aligned} & \Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\ &= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \dots \vee u_n \text{ not covered}] \\ &\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \leq ne^{-\ell} . \end{aligned}$$

Lemma 69

With high probability $\mathcal{O}(\log n)$ rounds suffice.

$$\begin{aligned}
& \Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\
&= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \dots \vee u_n \text{ not covered}] \\
&\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \leq ne^{-\ell} .
\end{aligned}$$

Lemma 69

With high probability $\mathcal{O}(\log n)$ rounds suffice.

With high probability:

For any constant α the number of rounds is at most $\mathcal{O}(\log n)$ with probability at least $1 - n^{-\alpha}$.

Proof: We have

$$\Pr[\text{\#rounds} \geq (\alpha + 1) \ln n] \leq n e^{-(\alpha+1) \ln n} = n^{-\alpha} .$$

Expected Cost

- ▶ Version A.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u .

Expected Cost

- ▶ Version A.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u .

$E[\text{cost}]$

Expected Cost

► Version A.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u .

$$E[\text{cost}] \leq (\alpha + 1) \ln n \cdot \text{cost}(LP) + (n \cdot \text{OPT}) n^{-\alpha}$$

Expected Cost

► Version A.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u .

$$E[\text{cost}] \leq (\alpha + 1) \ln n \cdot \text{cost}(LP) + (n \cdot \text{OPT}) n^{-\alpha} = \mathcal{O}(\ln n) \cdot \text{OPT}$$

Expected Cost

- ▶ Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] =$$

Expected Cost

- ▶ Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$\begin{aligned} E[\text{cost}] &= \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \end{aligned}$$

Expected Cost

- ▶ Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$E[\text{cost} \mid \text{success}]$$

Expected Cost

► Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$E[\text{cost} \mid \text{success}] \\ = \frac{1}{\Pr[\text{succ.}]} \left(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \right)$$

Expected Cost

► Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$\begin{aligned} E[\text{cost}] &= \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \end{aligned}$$

This means

$$\begin{aligned} E[\text{cost} \mid \text{success}] &= \frac{1}{\Pr[\text{succ.}]} \left(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \right) \\ &\leq \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \leq \frac{1}{1 - n^{-\alpha}} (\alpha + 1) \ln n \cdot \text{cost}(\text{LP}) \end{aligned}$$

Expected Cost

► Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$\begin{aligned} E[\text{cost}] &= \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \end{aligned}$$

This means

$$\begin{aligned} E[\text{cost} \mid \text{success}] &= \frac{1}{\Pr[\text{succ.}]} \left(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \right) \\ &\leq \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \leq \frac{1}{1 - n^{-\alpha}} (\alpha + 1) \ln n \cdot \text{cost}(\text{LP}) \\ &\leq 2(\alpha + 1) \ln n \cdot \text{OPT} \end{aligned}$$

Expected Cost

► Version B.

Repeat for $s = (\alpha + 1) \ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$\begin{aligned} E[\text{cost}] &= \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \end{aligned}$$

This means

$$\begin{aligned} E[\text{cost} \mid \text{success}] &= \frac{1}{\Pr[\text{succ.}]} \left(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \right) \\ &\leq \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \leq \frac{1}{1 - n^{-\alpha}} (\alpha + 1) \ln n \cdot \text{cost}(\text{LP}) \\ &\leq 2(\alpha + 1) \ln n \cdot \text{OPT} \end{aligned}$$

for $n \geq 2$ and $\alpha \geq 1$.

Randomized rounding gives an $\mathcal{O}(\log n)$ approximation. The running time is polynomial with high probability.

Theorem 70 (without proof)

There is no approximation algorithm for set cover with approximation guarantee better than $\frac{1}{2} \log n$ unless NP has quasi-polynomial time algorithms (algorithms with running time $2^{\text{poly}(\log n)}$).

Randomized rounding gives an $\mathcal{O}(\log n)$ approximation. The running time is polynomial with high probability.

Theorem 70 (without proof)

There is no approximation algorithm for set cover with approximation guarantee better than $\frac{1}{2} \log n$ unless NP has quasi-polynomial time algorithms (algorithms with running time $2^{\text{poly}(\log n)}$).

Integrality Gap

The **integrality gap** of the SetCover LP is $\Omega(\log n)$.

- ▶ $n = 2^k - 1$
- ▶ Elements are all vectors $\vec{x}$ over $GF[2]$ of length k (excluding zero vector).
- ▶ Every vector $\vec{y}$ defines a set as follows

$$S_{\vec{y}} := \{\vec{x} \mid \vec{x}^T \vec{y} = 1\}$$

- ▶ each set contains 2^{k-1} vectors; each vector is contained in 2^{k-1} sets
- ▶ $x_i = \frac{1}{2^{k-1}} = \frac{2}{n+1}$ is fractional solution.

Integrality Gap

Every collection of $p < k$ sets does not cover all elements.

Hence, we get a gap of $\Omega(\log n)$.

Techniques:

- ▶ Deterministic Rounding
- ▶ Rounding of the Dual
- ▶ Primal Dual
- ▶ Greedy
- ▶ Randomized Rounding
- ▶ Local Search
- ▶ Rounding Data + Dynamic Programming

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j .
Schedule the jobs on m identical parallel machines such that the **Makespan** (finishing time of the last job) is minimized.

$$\begin{array}{ll}\min & L \\ \text{s.t.} & \forall \text{ machines } i \quad \sum_j p_j \cdot x_{j,i} \leq L \\ & \forall \text{ jobs } j \quad \sum_i x_{j,i} \geq 1 \\ & \forall i, j \quad x_{j,i} \in \{0, 1\}\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i .

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \dots, n\}$ has processing time p_j .
Schedule the jobs on m identical parallel machines such that the **Makespan** (finishing time of the last job) is minimized.

$$\begin{array}{ll}\min & L \\ \text{s.t.} & \forall \text{ machines } i \quad \sum_j p_j \cdot x_{j,i} \leq L \\ & \forall \text{ jobs } j \quad \sum_i x_{j,i} \geq 1 \\ & \forall i, j \quad x_{j,i} \in \{0, 1\}\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i .

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j , and let $C_{\max}$ be the makespan.

Let $C_{\max}^*$ denote the makespan of an optimal solution.

Clearly

$$C_{\max}^* \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j , and let $C_{\max}$ be the makespan.

Let $C_{\max}^*$ denote the makespan of an optimal solution.

Clearly

$$C_{\max}^* \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j , and let $C_{\max}$ be the makespan.

Let $C_{\max}^*$ denote the makespan of an optimal solution.

Clearly

$$C_{\max}^* \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m} \sum_j p_j$.

Therefore,

$$C_{\max}^* \geq \frac{1}{m} \sum_j p_j$$

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m} \sum_j p_j$.
Therefore,

$$C_{\max}^* \geq \frac{1}{m} \sum_j p_j$$

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ , because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ , because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ , because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let s_ℓ be its start time, and let c_ℓ be its completion time.

Note that every machine is busy before time s_ℓ , because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

Hence, the length of the schedule is at most

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

Hence, the length of the schedule is at most

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

Hence, the length of the schedule is at most

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

Hence, the length of the schedule is at most

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m} \sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m} \sum_j p_j \leq (2 - \frac{1}{m})C_{\max}^*$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m} \sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m} \sum_j p_j \leq (2 - \frac{1}{m})C_{\max}^*$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ .

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C_{\max}^*$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j .$$

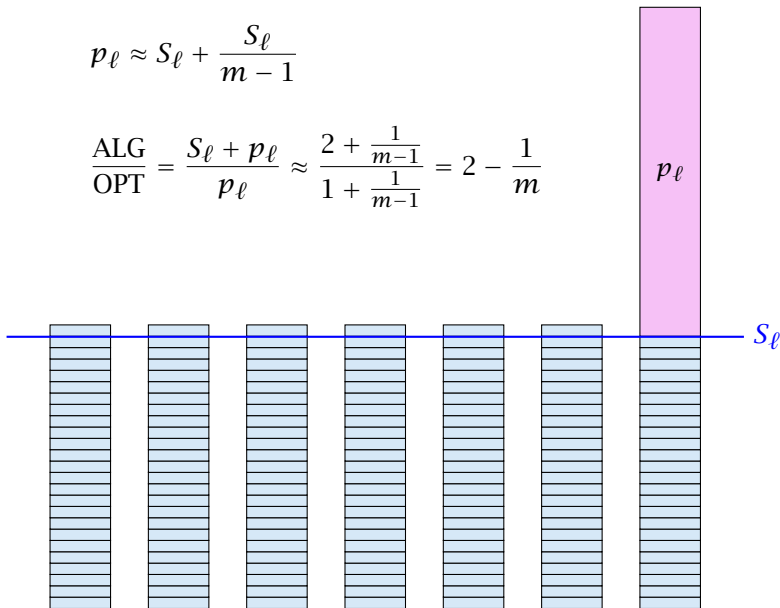
Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m} \sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m} \sum_j p_j \leq (2 - \frac{1}{m})C_{\max}^*$$

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\text{ALG}}{\text{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



A Greedy Strategy

List Scheduling:

Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:

Consider processes in some order. Assign the i -th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimality condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

List Scheduling:

Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:

Consider processes in some order. Assign the i -th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimality condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

List Scheduling:

Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:

Consider processes in some order. Assign the i -th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimality condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

List Scheduling:

Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:

Consider processes in some order. Assign the i -th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimality condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

Lemma 71

If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- ▶ Let $p_1 \geq \dots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- ▶ Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- ▶ If $p_n \leq C_{\max}^*/3$ the previous analysis gives us a schedule length of at most

$$C_{\max}^* + p_n \leq \frac{4}{3} C_{\max}^* .$$

Fence:

This means that all jobs must have a processing time

greater than $C_{\max}^*/3$.

But then any machine in the optimum schedule can handle at most two jobs.

For such instances, $\frac{4}{3} C_{\max}^*$ is a tight bound. This is optimal!

Proof:

- ▶ Let $p_1 \geq \dots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- ▶ Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- ▶ If $p_n \leq C_{\max}^*/3$ the previous analysis gives us a schedule length of at most

$$C_{\max}^* + p_n \leq \frac{4}{3} C_{\max}^* .$$

Example: $p_1 = 10, p_2 = 9, p_3 = 8, p_4 = 7, p_5 = 6, p_6 = 5, p_7 = 4, p_8 = 3, p_9 = 2, p_{10} = 1$.
This means that all jobs must have a processing time of at least 3.

Example: $p_1 = 10, p_2 = 9, p_3 = 8, p_4 = 7, p_5 = 6, p_6 = 5, p_7 = 4, p_8 = 3, p_9 = 2, p_{10} = 1$.

But the greedy algorithm in the algorithm schedule can handle a job with a processing time of 1.

Example: $p_1 = 10, p_2 = 9, p_3 = 8, p_4 = 7, p_5 = 6, p_6 = 5, p_7 = 4, p_8 = 3, p_9 = 2, p_{10} = 1$.

Proof:

- ▶ Let $p_1 \geq \dots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- ▶ Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- ▶ If $p_n \leq C_{\max}^*/3$ the previous analysis gives us a schedule length of at most

$$C_{\max}^* + p_n \leq \frac{4}{3}C_{\max}^* .$$

Hence, $p_n > C_{\max}^*/3$.

This means that all jobs have a processing time at most $2p_n$.
For every $j \in \{1, \dots, n\}$ we have $p_j \leq 2p_n$.
Hence, $C_{\max}^* \leq 2p_n$.
Hence, $p_n \geq C_{\max}^*/2$.

Proof:

- ▶ Let $p_1 \geq \dots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- ▶ Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- ▶ If $p_n \leq C_{\max}^*/3$ the previous analysis gives us a schedule length of at most

$$C_{\max}^* + p_n \leq \frac{4}{3}C_{\max}^* .$$

Hence, $p_n > C_{\max}^*/3$.

- ▶ This means that all jobs must have a processing time $> C_{\max}^*/3$.
- ▶ But then any machine in the optimum schedule can handle at most two jobs.
- ▶ For such instances Longest-Processing-Time-First is optimal.

Proof:

- ▶ Let $p_1 \geq \dots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- ▶ Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- ▶ If $p_n \leq C_{\max}^*/3$ the previous analysis gives us a schedule length of at most

$$C_{\max}^* + p_n \leq \frac{4}{3}C_{\max}^* .$$

Hence, $p_n > C_{\max}^*/3$.

- ▶ This means that all jobs must have a processing time $> C_{\max}^*/3$.
- ▶ But then any machine in the optimum schedule can handle at most two jobs.
- ▶ For such instances Longest-Processing-Time-First is optimal.

Proof:

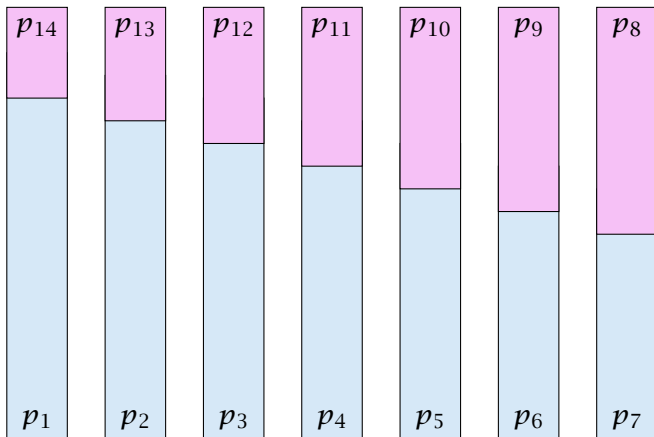
- ▶ Let $p_1 \geq \dots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- ▶ Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- ▶ If $p_n \leq C_{\max}^*/3$ the previous analysis gives us a schedule length of at most

$$C_{\max}^* + p_n \leq \frac{4}{3} C_{\max}^* .$$

Hence, $p_n > C_{\max}^*/3$.

- ▶ This means that all jobs must have a processing time $> C_{\max}^*/3$.
- ▶ But then any machine in the optimum schedule can handle at most two jobs.
- ▶ For such instances Longest-Processing-Time-First is optimal.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



- ▶ We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- ▶ If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B .
- ▶ Let p_A and p_B be the other job scheduled on A and B , respectively.
- ▶ $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- ▶ Repeat the above argument for the remaining machines.

- ▶ We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- ▶ If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B .
- ▶ Let p_A and p_B be the other job scheduled on A and B , respectively.
- ▶ $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- ▶ Repeat the above argument for the remaining machines.

- ▶ We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- ▶ If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B .
- ▶ Let p_A and p_B be the other job scheduled on A and B , respectively.
- ▶ $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- ▶ Repeat the above argument for the remaining machines.

- ▶ We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- ▶ If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B .
- ▶ Let p_A and p_B be the other job scheduled on A and B , respectively.
- ▶ $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- ▶ Repeat the above argument for the remaining machines.

- ▶ We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- ▶ If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B .
- ▶ Let p_A and p_B be the other job scheduled on A and B , respectively.
- ▶ $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- ▶ Repeat the above argument for the remaining machines.

Tight Example

- ▶ $2m + 1$ jobs



Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)



Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



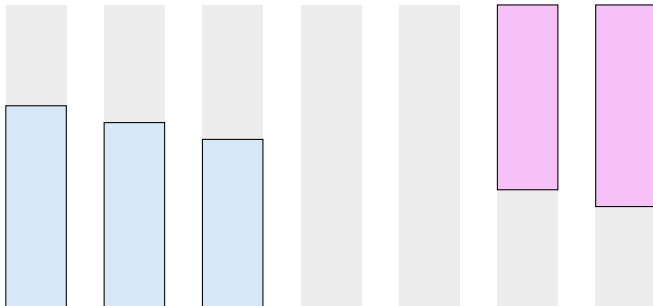
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



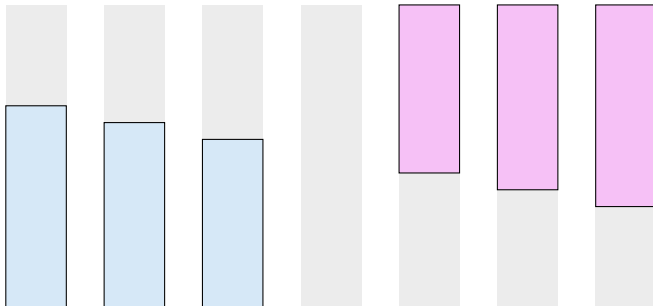
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



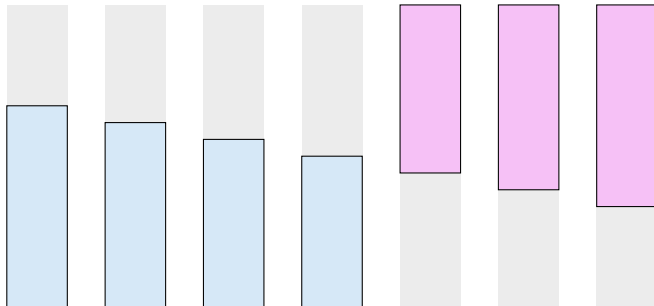
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



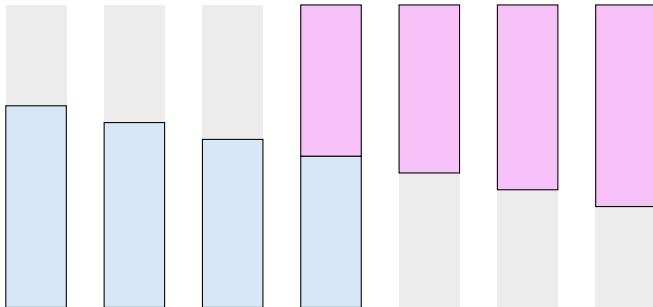
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



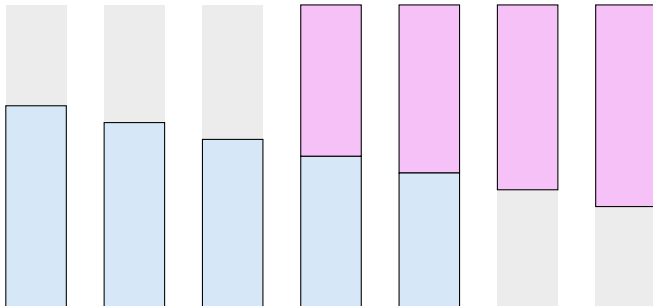
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



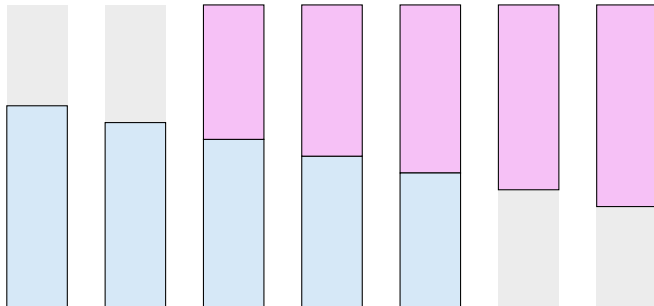
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



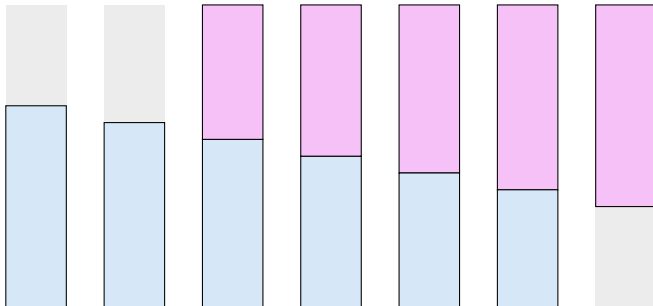
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



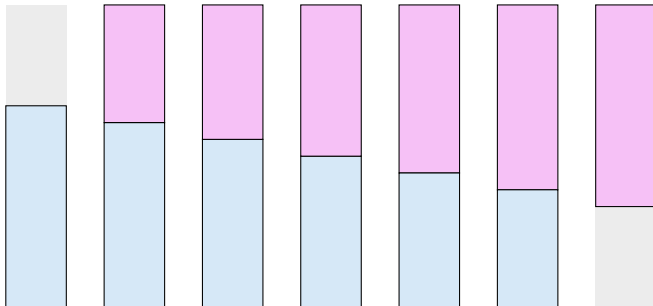
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



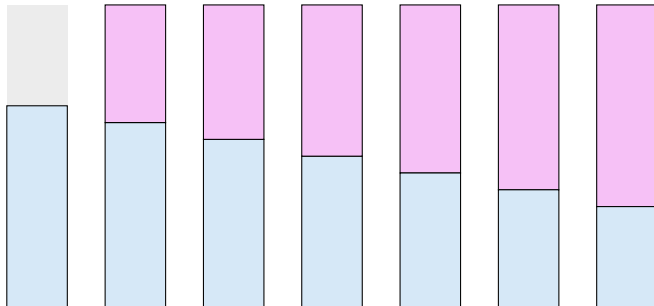
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



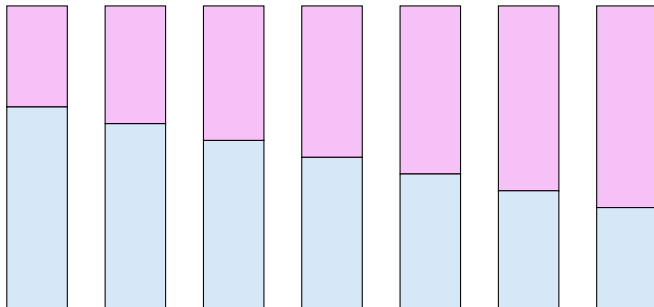
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



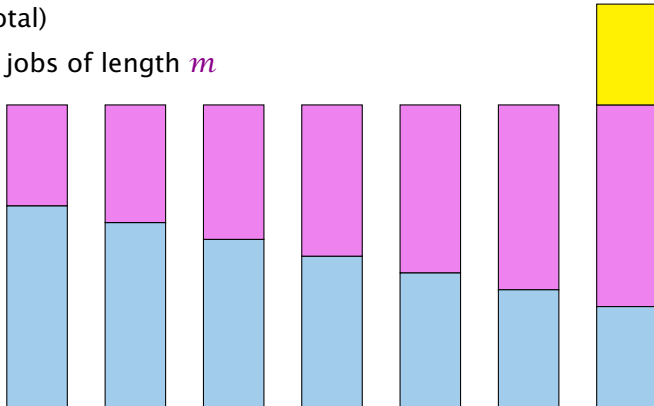
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



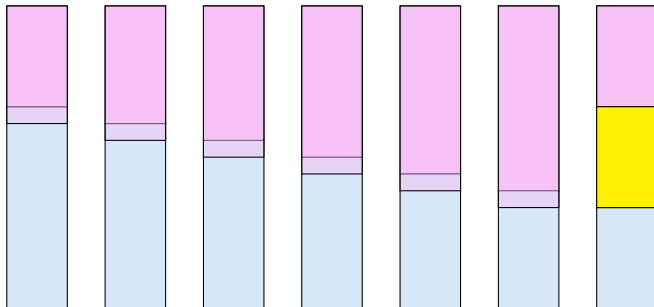
Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



Tight Example

- ▶ $2m + 1$ jobs
- ▶ 2 jobs with length $2m, 2m - 2, \dots, m + 1$ ($2m - 2$ jobs in total)
- ▶ 3 jobs of length m



14 Rounding Data + Dynamic Programming

Knapsack:

Given a set of items $\{1, \dots, n\}$, where the i -th item has weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, and given a threshold W . Find a subset $I \subseteq \{1, \dots, n\}$ of items of total weight at most W such that the profit is maximized (we can assume each $w_i \leq W$).

$$\begin{array}{ll} \max & \sum_{i=1}^n p_i x_i \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \leq W \\ & \forall i \in \{1, \dots, n\} \quad x_i \in \{0, 1\} \end{array}$$

14 Rounding Data + Dynamic Programming

Knapsack:

Given a set of items $\{1, \dots, n\}$, where the i -th item has weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, and given a threshold W . Find a subset $I \subseteq \{1, \dots, n\}$ of items of total weight at most W such that the profit is maximized (we can assume each $w_i \leq W$).

$$\begin{array}{ll} \max & \sum_{i=1}^n p_i x_i \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \leq W \\ & \forall i \in \{1, \dots, n\} \quad x_i \in \{0, 1\} \end{array}$$

14 Rounding Data + Dynamic Programming

Algorithm 1 Knapsack

```
1:  $A(1) \leftarrow [(0, 0), (p_1, w_1)]$ 
2: for  $j \leftarrow 2$  to  $n$  do
3:    $A(j) \leftarrow A(j - 1)$ 
4:   for each  $(p, w) \in A(j - 1)$  do
5:     if  $w + w_j \leq W$  then
6:       add  $(p + p_j, w + w_j)$  to  $A(j)$ 
7:       remove dominated pairs from  $A(j)$ 
8: return  $\max_{(p, w) \in A(n)} p$ 
```

The running time is $\mathcal{O}(n \cdot \min\{W, P\})$, where $P = \sum_i p_i$ is the total profit of all items. This is only **pseudo-polynomial**.

14 Rounding Data + Dynamic Programming

Definition 72

An algorithm is said to have pseudo-polynomial running time if the running time is polynomial when the numerical part of the input is encoded in unary.

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.
- ▶ Set $p'_i := \lfloor p_i / \mu \rfloor$ for all i .

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.
- ▶ Set $p'_i := \lfloor p_i / \mu \rfloor$ for all i .
- ▶ Run the dynamic programming algorithm on this revised instance.

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.
- ▶ Set $p'_i := \lfloor p_i / \mu \rfloor$ for all i .
- ▶ Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP')$$

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.
- ▶ Set $p'_i := \lfloor p_i / \mu \rfloor$ for all i .
- ▶ Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\left(n \sum_i p'_i\right)$$

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.
- ▶ Set $p'_i := \lfloor p_i / \mu \rfloor$ for all i .
- ▶ Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\left(n \sum_i p'_i\right) = \mathcal{O}\left(n \sum_i \left\lfloor \frac{p_i}{\epsilon M / n} \right\rfloor\right)$$

14 Rounding Data + Dynamic Programming

- ▶ Let M be the maximum profit of an element.
- ▶ Set $\mu := \epsilon M / n$.
- ▶ Set $p'_i := \lfloor p_i / \mu \rfloor$ for all i .
- ▶ Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\left(n \sum_i p'_i\right) = \mathcal{O}\left(n \sum_i \left\lfloor \frac{p_i}{\epsilon M / n} \right\rfloor\right) \leq \mathcal{O}\left(\frac{n^3}{\epsilon}\right) .$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\sum_{i \in S} p_i$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\sum_{i \in S} p_i \geq \mu \sum_{i \in S} p'_i$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i \in S} p_i &\geq \mu \sum_{i \in S} p'_i \\ &\geq \mu \sum_{i \in O} p'_i\end{aligned}$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i \in S} p_i &\geq \mu \sum_{i \in S} p'_i \\ &\geq \mu \sum_{i \in O} p'_i \\ &\geq \sum_{i \in O} p_i - |O|\mu\end{aligned}$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i \in S} p_i &\geq \mu \sum_{i \in S} p'_i \\ &\geq \mu \sum_{i \in O} p'_i \\ &\geq \sum_{i \in O} p_i - |O|\mu \\ &\geq \sum_{i \in O} p_i - n\mu\end{aligned}$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i \in S} p_i &\geq \mu \sum_{i \in S} p'_i \\ &\geq \mu \sum_{i \in O} p'_i \\ &\geq \sum_{i \in O} p_i - |O|\mu \\ &\geq \sum_{i \in O} p_i - n\mu \\ &= \sum_{i \in O} p_i - \epsilon M\end{aligned}$$

14 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i \in S} p_i &\geq \mu \sum_{i \in S} p'_i \\ &\geq \mu \sum_{i \in O} p'_i \\ &\geq \sum_{i \in O} p_i - |O|\mu \\ &\geq \sum_{i \in O} p_i - n\mu \\ &= \sum_{i \in O} p_i - \epsilon M \\ &\geq (1 - \epsilon) \text{OPT} .\end{aligned}$$

Scheduling Revisited

The previous analysis of the scheduling algorithm gave a makespan of

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where ℓ is the last job to complete.

Scheduling Revisited

The previous analysis of the scheduling algorithm gave a makespan of

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where ℓ is the last job to complete.

Together with the observation that if each $p_i \geq \frac{1}{3}C_{\max}^*$ then LPT is optimal this gave a $4/3$ -approximation.

14.2 Scheduling Revisited

Partition the input into **long** jobs and **short** jobs.

14.2 Scheduling Revisited

Partition the input into **long** jobs and **short** jobs.

A job j is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

14.2 Scheduling Revisited

Partition the input into **long** jobs and **short** jobs.

A job j is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

Idea:

1. Find the optimum Makespan for the long jobs by brute force.

14.2 Scheduling Revisited

Partition the input into **long** jobs and **short** jobs.

A job j is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

Idea:

1. Find the optimum Makespan for the long jobs by brute force.
2. Then use the list scheduling algorithm for the short jobs, always assigning the next job to the least loaded machine.

We still have a cost of

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where ℓ is the last job (this only requires that all machines are busy before time S_ℓ).

We still have a cost of

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where ℓ is the last job (this only requires that all machines are busy before time S_ℓ).

If ℓ is a long job, then the schedule must be optimal, as it consists of an optimal schedule of long jobs plus a schedule for short jobs.

We still have a cost of

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where ℓ is the last job (this only requires that all machines are busy before time S_ℓ).

If ℓ is a long job, then the schedule must be optimal, as it consists of an optimal schedule of long jobs plus a schedule for short jobs.

If ℓ is a short job its length is at most

$$p_\ell \leq \sum_j p_j / (mk)$$

which is at most $C_{\max}^* / k$.

Hence we get a schedule of length at most

$$\left(1 + \frac{1}{k}\right) C_{\max}^*$$

There are at most km long jobs. Hence, the number of possibilities of scheduling these jobs on m machines is at most m^{km} , which is constant if m is constant. Hence, it is easy to implement the algorithm in polynomial time.

Theorem 73

The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling n jobs on m identical machines if m is constant.

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

Hence we get a schedule of length at most

$$\left(1 + \frac{1}{k}\right) C_{\max}^*$$

There are at most km long jobs. Hence, the number of possibilities of scheduling these jobs on m machines is at most m^{km} , which is constant if m is constant. Hence, it is easy to implement the algorithm in polynomial time.

Theorem 73

The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling n jobs on m identical machines if m is constant.

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

Hence we get a schedule of length at most

$$\left(1 + \frac{1}{k}\right) C_{\max}^*$$

There are at most km long jobs. Hence, the number of possibilities of scheduling these jobs on m machines is at most m^{km} , which is constant if m is constant. Hence, it is easy to implement the algorithm in polynomial time.

Theorem 73

The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling n jobs on m identical machines if m is constant.

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:

On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m} \sum_j p_j$).

We partition the jobs into **long** jobs and **short** jobs:

- ▶ A job is long if its size is larger than T/k .
- ▶ Otw. it is a short job.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:

On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m} \sum_j p_j$).

We partition the jobs into **long** jobs and **short** jobs:

- ▶ A job is long if its size is larger than T/k .
- ▶ Otw. it is a short job.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:

On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m} \sum_j p_j$).

We partition the jobs into **long** jobs and **short** jobs:

- ▶ A job is long if its size is larger than T/k .
- ▶ Otw. it is a short job.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:

On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m} \sum_j p_j$).

We partition the jobs into **long** jobs and **short** jobs:

- ▶ A job is long if its size is larger than T/k .
- ▶ Otw. it is a short job.

- ▶ We round all **long jobs** down to multiples of T/k^2 .
- ▶ For these rounded sizes we first find an optimal schedule.
- ▶ If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- ▶ If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- ▶ We round all **long jobs** down to multiples of T/k^2 .
- ▶ For these rounded sizes we first find an optimal schedule.
 - ▶ If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
 - ▶ If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- ▶ We round all **long jobs** down to multiples of T/k^2 .
- ▶ For these rounded sizes we first find an optimal schedule.
- ▶ If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- ▶ If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- ▶ We round all **long jobs** down to multiples of T/k^2 .
- ▶ For these rounded sizes we first find an optimal schedule.
- ▶ If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- ▶ If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most T .

There can be at most k (long) jobs assigned to a machine as otherwise their rounded sizes would add up to more than T (note that the rounded size of a long job is at least T/k).

Since, jobs had been rounded to multiples of T/k^2 going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1 + \frac{1}{k}\right)T .$$

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most T .

There can be at most k (long) jobs assigned to a machine as otherwise their rounded sizes would add up to more than T (note that the rounded size of a long job is at least T/k).

Since, jobs had been rounded to multiples of T/k^2 going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1 + \frac{1}{k}\right)T.$$

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most T .

There can be at most k (long) jobs assigned to a machine as otherwise their rounded sizes would add up to more than T (note that the rounded size of a long job is at least T/k).

Since, jobs had been rounded to multiples of T/k^2 going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1 + \frac{1}{k}\right)T .$$

During the second phase there always must exist a machine with load at most T , since T is larger than the average load.

Assigning the current (short) job to such a machine gives that the new load is at most

$$T + \frac{T}{k} \leq \left(1 + \frac{1}{k}\right)T .$$

During the second phase there always must exist a machine with load at most T , since T is larger than the average load. Assigning the current (short) job to such a machine gives that the new load is at most

$$T + \frac{T}{k} \leq \left(1 + \frac{1}{k}\right) T .$$

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i -th entry describes the number of jobs of size $\frac{i}{k^2}T$). **This is polynomial.**

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i -th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x . There are only $(k+1)^{k^2}$ different vectors.

This means there are **a constant** number of different machine configurations.

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i -th entry describes the number of jobs of size $\frac{i}{k^2}T$). **This is polynomial.**

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i -th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x . There are only $(k+1)^{k^2}$ different vectors.

This means there are a **constant** number of different machine configurations.

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otherwise the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i -th entry describes the number of jobs of size $\frac{i}{k^2}T$). **This is polynomial.**

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i -th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x . There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i -th entry describes the number of jobs of size $\frac{i}{k^2}T$). **This is polynomial.**

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i -th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x . There are only $(k+1)^{k^2}$ different vectors.

This means there are **a constant** number of different machine configurations.

Let $\text{OPT}(n_1, \dots, n_{k^2})$ be the **number of machines** that are required to schedule input vector $(n_1, \dots, n_{k^2})$ with Makespan at most T .

If $\text{OPT}(n_1, \dots, n_{k^2}) \leq m$ we can schedule the input.

We have

$$\begin{aligned} & \text{OPT}(n_1, \dots, n_{k^2}) \\ &= \begin{cases} 0 & (n_1, \dots, n_{k^2}) = 0 \\ 1 + \min_{(s_1, \dots, s_{k^2}) \in C} \text{OPT}(n_1 - s_1, \dots, n_{k^2} - s_{k^2}) & (n_1, \dots, n_{k^2}) \geq 0 \\ \infty & \text{otw.} \end{cases} \end{aligned}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

Let $\text{OPT}(n_1, \dots, n_{k^2})$ be the **number of machines** that are required to schedule input vector $(n_1, \dots, n_{k^2})$ with Makespan at most T .

If $\text{OPT}(n_1, \dots, n_{k^2}) \leq m$ we can schedule the input.

We have

$$\text{OPT}(n_1, \dots, n_{k^2}) = \begin{cases} 0 & (n_1, \dots, n_{k^2}) = 0 \\ 1 + \min_{(s_1, \dots, s_{k^2}) \in C} \text{OPT}(n_1 - s_1, \dots, n_{k^2} - s_{k^2}) & (n_1, \dots, n_{k^2}) \not\geq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

Let $\text{OPT}(n_1, \dots, n_{k^2})$ be the **number of machines** that are required to schedule input vector $(n_1, \dots, n_{k^2})$ with Makespan at most T .

If $\text{OPT}(n_1, \dots, n_{k^2}) \leq m$ we can schedule the input.

We have

$$\text{OPT}(n_1, \dots, n_{k^2}) = \begin{cases} 0 & (n_1, \dots, n_{k^2}) = 0 \\ 1 + \min_{(s_1, \dots, s_{k^2}) \in C} \text{OPT}(n_1 - s_1, \dots, n_{k^2} - s_{k^2}) & (n_1, \dots, n_{k^2}) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

Let $\text{OPT}(n_1, \dots, n_{k^2})$ be the **number of machines** that are required to schedule input vector $(n_1, \dots, n_{k^2})$ with Makespan at most T .

If $\text{OPT}(n_1, \dots, n_{k^2}) \leq m$ we can schedule the input.

We have

$$\text{OPT}(n_1, \dots, n_{k^2}) = \begin{cases} 0 & (n_1, \dots, n_{k^2}) = 0 \\ 1 + \min_{(s_1, \dots, s_{k^2}) \in C} \text{OPT}(n_1 - s_1, \dots, n_{k^2} - s_{k^2}) & (n_1, \dots, n_{k^2}) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a **strongly NP-complete** problem.

Theorem 74

There is no FPTAS for problems that are strongly NP-hard.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a **strongly NP-complete** problem.

Theorem 74

There is no FPTAS for problems that are strongly NP-hard.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a **strongly NP-complete** problem.

Theorem 74

There is no FPTAS for problems that are strongly NP-hard.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a **strongly NP-complete** problem.

Theorem 74

There is no FPTAS for problems that are strongly NP-hard.

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$

- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$

- ▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless P=NP

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$

▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless $P=NP$

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$
- ▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless P=NP

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$
- ▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless $P=NP$

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$
- ▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless P=NP

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$
- ▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless $P=NP$

- ▶ Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- ▶ We set $k := \lceil 2nq(n) \rceil \geq 2 \text{ OPT}$
- ▶ Then

$$\text{ALG} \leq \left(1 + \frac{1}{k}\right) \text{OPT} \leq \text{OPT} + \frac{1}{2}$$

- ▶ But this means that the algorithm computes the optimal solution as the optimum is integral.
- ▶ This means we can solve problem instances if processing times are polynomially bounded
- ▶ Running time is $\mathcal{O}(\text{poly}(n, k)) = \mathcal{O}(\text{poly}(n))$
- ▶ For strongly NP-complete problems this is not possible unless $P=NP$

More General

Let $\text{OPT}(n_1, \dots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_A)$ with Makespan at most T (A : number of different sizes).

If $\text{OPT}(n_1, \dots, n_A) \leq m$ we can schedule the input.

$$\begin{aligned} & \text{OPT}(n_1, \dots, n_A) \\ &= \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min_{(s_1, \dots, s_A) \in C} \text{OPT}(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \geq 0 \\ \infty & \text{otw.} \end{cases} \end{aligned}$$

where C is the set of all configurations.

$|C| \leq (B+1)^A$, where B is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just n^A entries.

More General

Let $\text{OPT}(n_1, \dots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_A)$ with Makespan at most T (A : number of different sizes).

If $\text{OPT}(n_1, \dots, n_A) \leq m$ we can schedule the input.

$$\text{OPT}(n_1, \dots, n_A) = \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min_{(s_1, \dots, s_A) \in C} \text{OPT}(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \geq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

$|C| \leq (B+1)^A$, where B is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just n^A entries.

More General

Let $\text{OPT}(n_1, \dots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_A)$ with Makespan at most T (A : number of different sizes).

If $\text{OPT}(n_1, \dots, n_A) \leq m$ we can schedule the input.

$$\begin{aligned} &\text{OPT}(n_1, \dots, n_A) \\ &= \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min_{(s_1, \dots, s_A) \in C} \text{OPT}(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \neq 0 \\ \infty & \text{otw.} \end{cases} \end{aligned}$$

where C is the set of all configurations.

$|C| \leq (B + 1)^A$, where B is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B + 1)^A n^A)$ because the dynamic programming table has just n^A entries.

Bin Packing

Given n items with sizes $s_1, \dots, s_n$ where

$$1 > s_1 \geq \dots \geq s_n > 0 .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

Theorem 75

There is no ρ -approximation for Bin Packing with $\rho < 3/2$ unless $P = NP$.

Bin Packing

Given n items with sizes $s_1, \dots, s_n$ where

$$1 > s_1 \geq \dots \geq s_n > 0 .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

Theorem 75

There is no ρ -approximation for Bin Packing with $\rho < 3/2$ unless $P = NP$.

Bin Packing

Proof

- ▶ In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- ▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- ▶ A ρ -approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- ▶ Hence, such an algorithm can solve Partition.

Bin Packing

Proof

- ▶ In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- ▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- ▶ A ρ -approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- ▶ Hence, such an algorithm can solve Partition.

Bin Packing

Proof

- ▶ In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- ▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- ▶ A ρ -approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- ▶ Hence, such an algorithm can solve Partition.

Bin Packing

Proof

- ▶ In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- ▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- ▶ A ρ -approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- ▶ Hence, such an algorithm can solve Partition.

Definition 76

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1 + \epsilon)\text{OPT} + c$ for minimization problems.

Definition 76

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1 + \epsilon)\text{OPT} + c$ for minimization problems.

- ▶ Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.
- ▶ However, we will develop an APTAS for Bin Packing.

Definition 76

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1 + \epsilon)\text{OPT} + c$ for minimization problems.

- ▶ Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.
- ▶ However, we will develop an APTAS for Bin Packing.

Bin Packing

Again we can differentiate between small and large items.

Lemma 77

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

Bin Packing

Again we can differentiate between small and large items.

Lemma 77

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- ▶ If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.
- ▶ Hence, $r(1 - \gamma) \leq \text{SIZE}(I)$ where r is the number of nearly-full bins.
- ▶ This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 77

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- ▶ If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.
- ▶ Hence, $r(1 - \gamma) \leq \text{SIZE}(I)$ where r is the number of nearly-full bins.
- ▶ This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 77

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- ▶ If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.
- ▶ Hence, $r(1 - \gamma) \leq \text{SIZE}(I)$ where r is the number of nearly-full bins.
- ▶ This gives the lemma.

Choose $\gamma = \epsilon/2$. Then we either use ℓ bins or at most

$$\frac{1}{1 - \epsilon/2} \cdot \text{OPT} + 1 \leq (1 + \epsilon) \cdot \text{OPT} + 1$$

bins.

It remains to find an algorithm for the large items.

Linear Grouping:

Generate an instance I' (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

Linear Grouping:

Generate an instance I' (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

Linear Grouping:

Generate an instance I' (for large items) as follows.

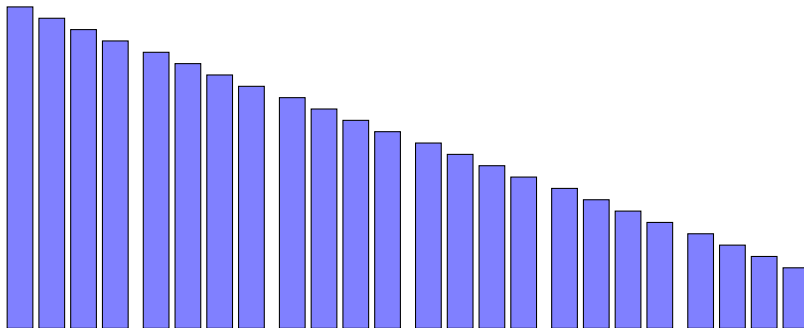
- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

Linear Grouping:

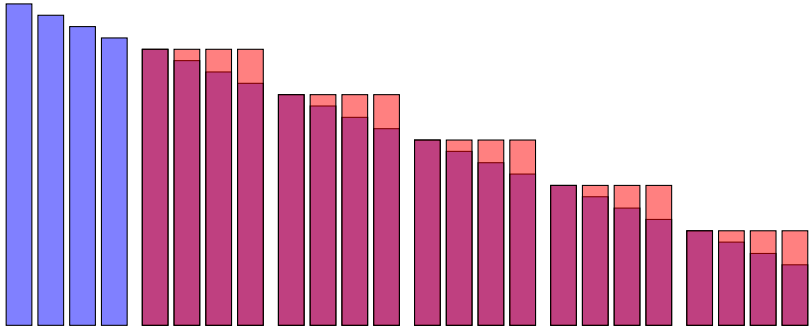
Generate an instance I' (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

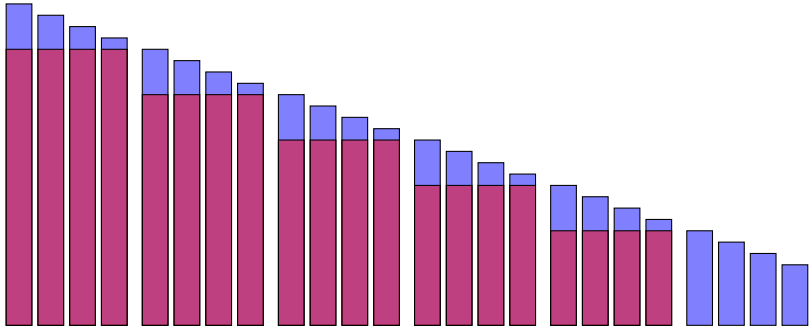
Linear Grouping



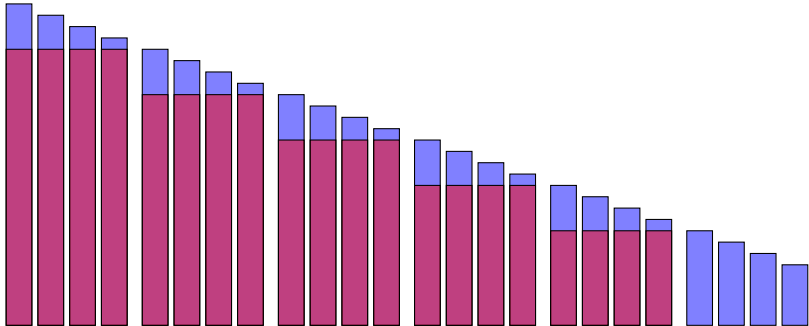
Linear Grouping



Linear Grouping



Linear Grouping



Lemma 78

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 1:

Any bin packing for I gives a bin packing for I' as follows:

1. Pack the items of group 1, where in the packing for I the items for group 1 have been packed.

2. Pack the items of groups 2, where in the packing for I the items for group 2 have been packed.

Lemma 78

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 1:

- ▶ Any bin packing for I gives a bin packing for I' as follows.
 - ▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
 - ▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
 - ▶ ...

Lemma 78

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 1:

- ▶ Any bin packing for I gives a bin packing for I' as follows.
- ▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- ▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ▶ ...

Lemma 78

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 1:

- ▶ Any bin packing for I gives a bin packing for I' as follows.
- ▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- ▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;



Lemma 78

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 1:

- ▶ Any bin packing for I gives a bin packing for I' as follows.
- ▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- ▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ▶ ...

Lemma 79

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 2:

- ▶ Any bin packing for I' gives a bin packing for I as follows.
- ▶ Pack the items of group 1 into k new bins;
- ▶ Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ▶ ...

Lemma 79

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 2:

- ▶ Any bin packing for I' gives a bin packing for I as follows.
- ▶ Pack the items of group 1 into k new bins;
- ▶ Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ▶ ...

Lemma 79

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 2:

- ▶ Any bin packing for I' gives a bin packing for I as follows.
- ▶ Pack the items of group 1 into k new bins;
- ▶ Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ▶ ...

Lemma 79

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

Proof 2:

- ▶ Any bin packing for I' gives a bin packing for I as follows.
- ▶ Pack the items of group 1 into k new bins;
- ▶ Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ▶ ...

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (note that $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- ▶ cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

- ▶ running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (note that $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- ▶ cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

- ▶ running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (note that $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- ▶ cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

- ▶ running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (note that $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

► cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

► running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (note that $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

► cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

► running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (note that $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- ▶ cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

- ▶ running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\text{OPT}(I) + \mathcal{O}(\log^2(\text{SIZE}(I))) .$$

Note that this is usually better than a guarantee of

$$(1 + \epsilon)\text{OPT}(I) + 1 .$$

Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\text{OPT}(I) + \mathcal{O}(\log^2(\text{SIZE}(I))) .$$

Note that this is usually better than a guarantee of

$$(1 + \epsilon)\text{OPT}(I) + 1 .$$

Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\text{OPT}(I) + \mathcal{O}(\log^2(\text{SIZE}(I))) .$$

Note that this is usually better than a guarantee of

$$(1 + \epsilon)\text{OPT}(I) + 1 .$$

Change of Notation:

- ▶ Group pieces of identical size.
- ▶ Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1 .
- ▶ s_2 is second largest size and b_2 number of pieces of size s_2 ;
- ▶ ...
- ▶ s_m smallest size and b_m number of pieces of size s_m .

Change of Notation:

- ▶ Group pieces of identical size.
- ▶ Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1 .
- ▶ s_2 is second largest size and b_2 number of pieces of size s_2 ;
- ▶ ...
- ▶ s_m smallest size and b_m number of pieces of size s_m .

Configuration LP

Change of Notation:

- ▶ Group pieces of identical size.
- ▶ Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1 .
- ▶ s_2 is second largest size and b_2 number of pieces of size s_2 ;
- ▶ ...
- ▶ s_m smallest size and b_m number of pieces of size s_m .

Configuration LP

Change of Notation:

- ▶ Group pieces of identical size.
- ▶ Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1 .
- ▶ s_2 is second largest size and b_2 number of pieces of size s_2 ;
- ▶ ...
- ▶ s_m smallest size and b_m number of pieces of size s_m .

Change of Notation:

- ▶ Group pieces of identical size.
- ▶ Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1 .
- ▶ s_2 is second largest size and b_2 number of pieces of size s_2 ;
- ▶ ...
- ▶ s_m smallest size and b_m number of pieces of size s_m .

Configuration LP

A possible packing of a bin can be described by an m -tuple $(t_1, \dots, t_m)$, where t_i describes the number of pieces of size s_i .

Clearly,

$$\sum_i t_i \cdot s_i \leq 1.$$

We call a vector that fulfills the above constraint a **configuration**.

Configuration LP

A possible packing of a bin can be described by an m -tuple $(t_1, \dots, t_m)$, where t_i describes the number of pieces of size s_i . Clearly,

$$\sum_i t_i \cdot s_i \leq 1 .$$

We call a vector that fulfills the above constraint a **configuration**.

Configuration LP

A possible packing of a bin can be described by an m -tuple $(t_1, \dots, t_m)$, where t_i describes the number of pieces of size s_i . Clearly,

$$\sum_i t_i \cdot s_i \leq 1 \ .$$

We call a vector that fulfills the above constraint a **configuration**.

Configuration LP

Let N be the number of configurations (**exponential**).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$\begin{array}{llll} \min & & \sum_{j=1}^N x_j & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^N T_{ji} x_j & \geq b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \geq 0 \\ & \forall j \in \{1, \dots, N\} & x_j & \text{integral} \end{array}$$

Configuration LP

Let N be the number of configurations (**exponential**).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \\ & \forall j \in \{1, \dots, N\} \quad x_j \text{ integral} \end{array}$$

Configuration LP

Let N be the number of configurations (**exponential**).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \\ & \forall j \in \{1, \dots, N\} \quad x_j \text{ integral} \end{array}$$

Configuration LP

Let N be the number of configurations (**exponential**).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$\begin{array}{llll} \min & & \sum_{j=1}^N x_j & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^N T_{ji} x_j & \geq b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \geq 0 \\ & \forall j \in \{1, \dots, N\} & x_j & \text{integral} \end{array}$$

How to solve this LP?

later...

We can assume that each item has size at least $1/\text{SIZE}(I)$.

Harmonic Grouping

- ▶ Sort items according to size (monotonically decreasing).
- ▶ Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- ▶ I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- ▶ Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

- ▶ Sort items according to size (monotonically decreasing).
- ▶ Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- ▶ I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- ▶ Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

- ▶ Sort items according to size (monotonically decreasing).
- ▶ Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- ▶ I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- ▶ Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

- ▶ Sort items according to size (monotonically decreasing).
- ▶ Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- ▶ I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- ▶ Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- ▶ Round all items in a group to the size of the largest group member.
- ▶ Delete all items from group G_1 and G_r .
- ▶ For groups $G_2, \dots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- ▶ Observe that $n_i \geq n_{i-1}$.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- ▶ Round all items in a group to the size of the largest group member.
- ▶ Delete all items from group G_1 and G_r .
- ▶ For groups $G_2, \dots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- ▶ Observe that $n_i \geq n_{i-1}$.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- ▶ Round all items in a group to the size of the largest group member.
- ▶ Delete all items from group G_1 and G_r .
- ▶ For groups $G_2, \dots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- ▶ Observe that $n_i \geq n_{i-1}$.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- ▶ Round all items in a group to the size of the largest group member.
- ▶ Delete all items from group G_1 and G_r .
- ▶ For groups $G_2, \dots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- ▶ Observe that $n_i \geq n_{i-1}$.

Lemma 80

The number of different sizes in I' is at most $\text{SIZE}(I)/2$.

Lemma 80

The number of different sizes in I' is at most $\text{SIZE}(I)/2$.

- ▶ Each group that survives (recall that G_1 and G_r are deleted) has total size at least 2.
- ▶ Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- ▶ All items in a group have the same size in I' .

Lemma 80

The number of different sizes in I' is at most $\text{SIZE}(I)/2$.

- ▶ Each group that survives (recall that G_1 and G_r are deleted) has total size at least 2.
- ▶ Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- ▶ All items in a group have the same size in I' .

Lemma 80

The number of different sizes in I' is at most $\text{SIZE}(I)/2$.

- ▶ Each group that survives (recall that G_1 and G_r are deleted) has total size at least 2.
- ▶ Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- ▶ All items in a group have the same size in I' .

Lemma 81

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

Let $\text{SIZE}(I)$ be the total size of items in I , and let $\mathcal{G}$ be an ϵ -approximate group scheduling algorithm.

Consider a group of items that is deleted from $\mathcal{G}$. If it consists of k items, then the items of total size at most ϵk .

Since $\mathcal{G}$ is an ϵ -approximate group scheduling algorithm, the total size of items that are deleted from $\mathcal{G}$ is at most $\epsilon \cdot \text{SIZE}(I)$.

Since the average item size is ϵ , the total size of items that are deleted from $\mathcal{G}$ is at most $\epsilon \cdot \text{SIZE}(I)$.

Summing over all k items that are deleted from $\mathcal{G}$ gives a bound of at most $\epsilon \cdot \text{SIZE}(I)$.

Since ϵ is an arbitrary constant, we can choose ϵ to be $1/\log(\text{SIZE}(I))$.

Therefore, the total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

Lemma 81

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

- ▶ The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- ▶ Consider a group G_i that has strictly more items than G_{i-1} .
- ▶ It discards $n_i - n_{i-1}$ pieces of total size at most

$$3 \frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the average piece size is only $3/n_i$.

- ▶ Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I))) .$$

(note that $n_r \leq \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

Lemma 81

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

- ▶ The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- ▶ Consider a group G_i that has strictly more items than G_{i-1} .
- ▶ It discards $n_i - n_{i-1}$ pieces of total size at most

$$3 \frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the average piece size is only $3/n_i$.

- ▶ Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_r-1} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I))) .$$

(note that $n_r \leq \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

Lemma 81

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

- ▶ The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- ▶ Consider a group G_i that has strictly more items than G_{i-1} .
- ▶ It discards $n_i - n_{i-1}$ pieces of total size at most

$$3 \frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the average piece size is only $3/n_i$.

- ▶ Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_r-1} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I))) .$$

(note that $n_r \leq \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

Lemma 81

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

- ▶ The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- ▶ Consider a group G_i that has strictly more items than G_{i-1} .
- ▶ It discards $n_i - n_{i-1}$ pieces of total size at most

$$3 \frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the average piece size is only $3/n_i$.

- ▶ Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I))) .$$

(note that $n_r \leq \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

Algorithm 1 BinPack

- 1: **if** $\text{SIZE}(I) < 10$ **then**
- 2: pack remaining items greedily
- 3: Apply harmonic grouping to create instance I' ; pack discarded items in at most $\mathcal{O}(\log(\text{SIZE}(I)))$ bins.
- 4: Let x be optimal solution to configuration LP
- 5: Pack $\lfloor x_j \rfloor$ bins in configuration T_j for all j ; call the packed instance I_1 .
- 6: Let I_2 be remaining pieces from I'
- 7: Pack I_2 via $\text{BinPack}(I_2)$

$$\text{OPT}_{\text{LP}}(I_1) + \text{OPT}_{\text{LP}}(I_2) \leq \text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$$

Proof:

- Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$.
- $\text{OPT}_{\text{LP}}(I_1)$ is feasible solution for I' (even integrally).
- $\text{OPT}_{\text{LP}}(I_2)$ is feasible solution for I' .

$$\text{OPT}_{\text{LP}}(I_1) + \text{OPT}_{\text{LP}}(I_2) \leq \text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$$

Proof:

- ▶ Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$
- ▶ $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- ▶ $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2 .

$$\text{OPT}_{\text{LP}}(I_1) + \text{OPT}_{\text{LP}}(I_2) \leq \text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$$

Proof:

- ▶ Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$
- ▶ $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- ▶ $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2 .

$$\text{OPT}_{\text{LP}}(I_1) + \text{OPT}_{\text{LP}}(I_2) \leq \text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$$

Proof:

- ▶ Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$
- ▶ $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- ▶ $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2 .

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1 .
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most OPT_{LP} many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1 .
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most OPT_{LP} many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1 .
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most OPT_{LP} many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1 .
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most OPT_{LP} many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1 .
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most OPT_{LP} many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

We can show that $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$ in total.

We can show that $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$ in total.

- ▶ The number of non-zero entries in the solution to the configuration LP for I' is at most the number of constraints, which is the number of different sizes ($\leq \text{SIZE}(I)/2$).
- ▶ The total size of items in I_2 can be at most $\sum_{j=1}^N x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

We can show that $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$ in total.

- ▶ The number of non-zero entries in the solution to the configuration LP for I' is at most the number of constraints, which is the number of different sizes ($\leq \text{SIZE}(I)/2$).
- ▶ The total size of items in I_2 can be at most $\sum_{j=1}^N x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

In total we have b_i pieces of size s_i .

Primal

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

Dual

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

In total we have b_i pieces of size s_i .

Primal

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

Dual

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

In total we have b_i pieces of size s_i .

Primal

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

Dual

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

Separation Oracle

Suppose that I am given variable assignment y for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

is feasible, i.e.,

$$\sum_{i=1}^m T_{ji} w_i \leq W$$

$$\sum_{i=1}^m T_{ji} h_i \leq H$$

$$\sum_{i=1}^m T_{ji} l_i \leq L$$

$$\sum_{i=1}^m T_{ji} p_i y_i > 0$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

$$\sum_{i=1}^m T_{ji} p_i y_i > \sum_{i=1}^m p_i y_i$$

But this is the Knapsack problem.

Separation Oracle

Suppose that I am given variable assignment γ for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- ▶ is feasible, i.e.,

$$\sum_{i=1}^m T_{ji} \cdot s_i \leq 1 ,$$

- ▶ and has a large profit

$$\sum_{i=1}^m T_{ji} \gamma_i > 1$$

But this is the Knapsack problem.

Separation Oracle

Suppose that I am given variable assignment γ for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- ▶ is feasible, i.e.,

$$\sum_{i=1}^m T_{ji} \cdot s_i \leq 1 ,$$

- ▶ and has a large profit

$$\sum_{i=1}^m T_{ji} \gamma_i > 1$$

But this is the Knapsack problem.

Separation Oracle

Suppose that I am given variable assignment γ for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- ▶ is feasible, i.e.,

$$\sum_{i=1}^m T_{ji} \cdot s_i \leq 1 ,$$

- ▶ and has a large profit

$$\sum_{i=1}^m T_{ji} \gamma_i > 1$$

But this is the Knapsack problem.

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual'

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

Primal'

$$\begin{array}{ll} \min & (1 + \epsilon') \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual'

$$\begin{array}{ll}\max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0\end{array}$$

Primal'

$$\begin{array}{ll}\min & (1 + \epsilon') \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0\end{array}$$

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual'

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

Primal'

$$\begin{array}{ll} \min & (1 + \epsilon') \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual'

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

Primal'

$$\begin{array}{ll} \min & (1 + \epsilon') \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

1. The constraints used when computing z certify that the solution is feasible for z .

2. Suppose that we drop all unused constraints. In z , we will compute the same solution (maybe for z).

3. Let z' be the dual without unused constraints.

4. The dual to z' is z , where we have variables for which the corresponding dual constraint has not been used.

5. The optimum value for z' is at most $(1 + \epsilon')z$.

6. We can compute the corresponding solution in z' .

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing z **certify** that the solution is feasible for DUAL' .
- ▶ Suppose that we drop all unused constraints in DUAL . We will compute the same solution feasible for DUAL' .
- ▶ Let DUAL'' be DUAL without unused constraints.
- ▶ The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- ▶ The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
- ▶ We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing z **certify** that the solution is feasible for DUAL' .
- ▶ Suppose that we drop all unused constraints in DUAL . We will compute the same solution feasible for DUAL' .
 - ▶ Let DUAL'' be DUAL without unused constraints.
 - ▶ The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
 - ▶ The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
 - ▶ We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing z **certify** that the solution is feasible for DUAL' .
- ▶ Suppose that we drop all unused constraints in DUAL . We will compute the same solution feasible for DUAL' .
- ▶ Let DUAL'' be DUAL without unused constraints.
 - ▶ The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
 - ▶ The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
 - ▶ We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing z **certify** that the solution is feasible for DUAL' .
- ▶ Suppose that we drop all unused constraints in DUAL . We will compute the same solution feasible for DUAL' .
- ▶ Let DUAL'' be DUAL without unused constraints.
- ▶ The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- ▶ The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
- ▶ We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing z **certify** that the solution is feasible for DUAL' .
- ▶ Suppose that we drop all unused constraints in DUAL . We will compute the same solution feasible for DUAL' .
- ▶ Let DUAL'' be DUAL without unused constraints.
- ▶ The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- ▶ The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
- ▶ We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing z **certify** that the solution is feasible for DUAL' .
- ▶ Suppose that we drop all unused constraints in DUAL . We will compute the same solution feasible for DUAL' .
- ▶ Let DUAL'' be DUAL without unused constraints.
- ▶ The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- ▶ The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
- ▶ We can compute the corresponding solution in polytime.

This gives that overall we need at most

$$(1 + \epsilon') \text{OPT}_{\text{LP}}(I) + \mathcal{O}(\log^2(\text{SIZE}(I)))$$

bins.

We can choose $\epsilon' = \frac{1}{\text{OPT}}$ as $\text{OPT} \leq \# \text{items}$ and since we have a **fully polynomial time approximation scheme (FPTAS)** for knapsack.

This gives that overall we need at most

$$(1 + \epsilon') \text{OPT}_{\text{LP}}(I) + \mathcal{O}(\log^2(\text{SIZE}(I)))$$

bins.

We can choose $\epsilon' = \frac{1}{\text{OPT}}$ as $\text{OPT} \leq \# \text{items}$ and since we have a **fully polynomial time approximation scheme (FPTAS)** for knapsack.

15.1 MAXSAT

Problem definition:

- ▶ n Boolean variables
- ▶ m clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- ▶ Non-negative weight w_j for each clause C_j .
- ▶ Find an assignment of true/false to the variables such that the total weight of clauses that are **satisfied** is maximum.

15.1 MAXSAT

Problem definition:

- ▶ n Boolean variables
- ▶ m clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- ▶ Non-negative weight w_j for each clause C_j .
- ▶ Find an assignment of true/false to the variables such that the total weight of clauses that are **satisfied** is maximum.

15.1 MAXSAT

Problem definition:

- ▶ n Boolean variables
- ▶ m clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- ▶ Non-negative weight w_j for each clause C_j .
- ▶ Find an assignment of true/false to the variables such that the total weight of clauses that are **satisfied** is maximum.

15.1 MAXSAT

Problem definition:

- ▶ n Boolean variables
- ▶ m clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- ▶ Non-negative weight w_j for each clause C_j .
- ▶ Find an assignment of true/false to the variables such that the total weight of clauses that are **satisfied** is maximum.

15.1 MAXSAT

Terminology:

- ▶ A variable x_i and its negation $\bar{x}_i$ are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- ▶ We assume a clause does not contain x_i and $\bar{x}_i$ for any i .
- ▶ x_i is called a **positive literal** while the negation $\bar{x}_i$ is called a **negative literal**.
- ▶ For a given clause C_j the number of its literals is called its **length** or **size** and denoted with ℓ_j .
- ▶ Clauses of length one are called **unit clauses**.

15.1 MAXSAT

Terminology:

- ▶ A variable x_i and its negation $\bar{x}_i$ are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- ▶ We assume a clause does not contain x_i and $\bar{x}_i$ for any i .
- ▶ x_i is called a **positive literal** while the negation $\bar{x}_i$ is called a **negative literal**.
- ▶ For a given clause C_j the number of its literals is called its **length** or **size** and denoted with ℓ_j .
- ▶ Clauses of length one are called **unit clauses**.

15.1 MAXSAT

Terminology:

- ▶ A variable x_i and its negation $\bar{x}_i$ are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- ▶ We assume a clause does not contain x_i and $\bar{x}_i$ for any i .
- ▶ x_i is called a **positive literal** while the negation $\bar{x}_i$ is called a **negative literal**.
- ▶ For a given clause C_j the number of its literals is called its **length** or **size** and denoted with ℓ_j .
- ▶ Clauses of length one are called **unit clauses**.

15.1 MAXSAT

Terminology:

- ▶ A variable x_i and its negation $\bar{x}_i$ are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- ▶ We assume a clause does not contain x_i and $\bar{x}_i$ for any i .
- ▶ x_i is called a **positive literal** while the negation $\bar{x}_i$ is called a **negative literal**.
- ▶ For a given clause C_j the number of its literals is called its **length** or **size** and denoted with ℓ_j .
- ▶ Clauses of length one are called **unit clauses**.

15.1 MAXSAT

Terminology:

- ▶ A variable x_i and its negation $\bar{x}_i$ are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- ▶ We assume a clause does not contain x_i and $\bar{x}_i$ for any i .
- ▶ x_i is called a **positive literal** while the negation $\bar{x}_i$ is called a **negative literal**.
- ▶ For a given clause C_j the number of its literals is called its **length** or **size** and denoted with ℓ_j .
- ▶ Clauses of length one are called **unit clauses**.

15.1 MAXSAT

Terminology:

- ▶ A variable x_i and its negation $\bar{x}_i$ are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- ▶ We assume a clause does not contain x_i and $\bar{x}_i$ for any i .
- ▶ x_i is called a **positive literal** while the negation $\bar{x}_i$ is called a **negative literal**.
- ▶ For a given clause C_j the number of its literals is called its **length** or **size** and denoted with ℓ_j .
- ▶ Clauses of length one are called **unit clauses**.

MAXSAT: Flipping Coins

Set each x_i independently to **true** with probability $\frac{1}{2}$ (and, hence, to **false** with probability $\frac{1}{2}$, as well).

Define random variable X_j with

$$X_j = \begin{cases} 1 & \text{if } C_j \text{ satisfied} \\ 0 & \text{otw.} \end{cases}$$

Then the total weight W of satisfied clauses is given by

$$W = \sum_j w_j X_j$$

Define random variable X_j with

$$X_j = \begin{cases} 1 & \text{if } C_j \text{ satisfied} \\ 0 & \text{otw.} \end{cases}$$

Then the total weight W of satisfied clauses is given by

$$W = \sum_j w_j X_j$$

$$E[W]$$

$$E[W] = \sum_j w_j E[X_j]$$

$$\begin{aligned} E[W] &= \sum_j w_j E[X_j] \\ &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \end{aligned}$$

$$\begin{aligned} E[W] &= \sum_j w_j E[X_j] \\ &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\ &= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \end{aligned}$$

$$\begin{aligned} E[W] &= \sum_j w_j E[X_j] \\ &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\ &= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\ &\geq \frac{1}{2} \sum_j w_j \end{aligned}$$

$$\begin{aligned} E[W] &= \sum_j w_j E[X_j] \\ &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\ &= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\ &\geq \frac{1}{2} \sum_j w_j \\ &\geq \frac{1}{2} \text{OPT} \end{aligned}$$

MAXSAT: LP formulation

- ▶ Let for a clause C_j , P_j be the set of positive literals and N_j the set of negative literals.

$$C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \bar{x}_i$$

$$\begin{array}{ll} \max & \sum_j w_j z_j \\ \text{s.t.} & \forall j \quad \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \\ & \forall i \quad y_i \in \{0, 1\} \\ & \forall j \quad z_j \leq 1 \end{array}$$

MAXSAT: LP formulation

- ▶ Let for a clause C_j , P_j be the set of positive literals and N_j the set of negative literals.

$$C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \bar{x}_i$$

$$\begin{array}{ll} \max & \sum_j w_j z_j \\ \text{s.t.} & \forall j \quad \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \\ & \forall i \quad y_i \in \{0, 1\} \\ & \forall j \quad z_j \leq 1 \end{array}$$

MAXSAT: Randomized Rounding

Set each x_i independently to **true** with probability y_i (and, hence, to **false** with probability $(1 - y_i)$).

Lemma 82 (Geometric Mean $\leq$ Arithmetic Mean)

For any nonnegative $a_1, \dots, a_k$

$$\left(\prod_{i=1}^k a_i \right)^{1/k} \leq \frac{1}{k} \sum_{i=1}^k a_i$$

Definition 83

A function f on an interval I is **concave** if for any two points s and r from I and any $\lambda \in [0, 1]$ we have

$$f(\lambda s + (1 - \lambda)r) \geq \lambda f(s) + (1 - \lambda)f(r)$$

Lemma 84

Let f be a concave function on the interval $[0, 1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$f(\lambda)$$

for $\lambda \in [0, 1]$.

Definition 83

A function f on an interval I is **concave** if for any two points s and r from I and any $\lambda \in [0, 1]$ we have

$$f(\lambda s + (1 - \lambda)r) \geq \lambda f(s) + (1 - \lambda)f(r)$$

Lemma 84

Let f be a concave function on the interval $[0, 1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$\begin{aligned} f(\lambda) &= f((1 - \lambda)0 + \lambda 1) \\ &\geq (1 - \lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for $\lambda \in [0, 1]$.

Definition 83

A function f on an interval I is **concave** if for any two points s and r from I and any $\lambda \in [0, 1]$ we have

$$f(\lambda s + (1 - \lambda)r) \geq \lambda f(s) + (1 - \lambda)f(r)$$

Lemma 84

Let f be a concave function on the interval $[0, 1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$\begin{aligned} f(\lambda) &= f((1 - \lambda)0 + \lambda 1) \\ &\geq (1 - \lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for $\lambda \in [0, 1]$.

Definition 83

A function f on an interval I is **concave** if for any two points s and r from I and any $\lambda \in [0, 1]$ we have

$$f(\lambda s + (1 - \lambda)r) \geq \lambda f(s) + (1 - \lambda)f(r)$$

Lemma 84

Let f be a concave function on the interval $[0, 1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$\begin{aligned} f(\lambda) &= f((1 - \lambda)0 + \lambda 1) \\ &\geq (1 - \lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for $\lambda \in [0, 1]$.

$\Pr[C_j \text{ not satisfied}]$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i$$

$$\begin{aligned}\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i \\ &\leq \left[\frac{1}{\ell_j} \left(\sum_{i \in P_j} (1 - y_i) + \sum_{i \in N_j} y_i \right) \right]^{\ell_j}\end{aligned}$$

$$\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i \\
&\leq \left[\frac{1}{\ell_j} \left(\sum_{i \in P_j} (1 - y_i) + \sum_{i \in N_j} y_i \right) \right]^{\ell_j} \\
&= \left[1 - \frac{1}{\ell_j} \left(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \right) \right]^{\ell_j}
\end{aligned}$$

$$\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i \\
&\leq \left[\frac{1}{\ell_j} \left(\sum_{i \in P_j} (1 - y_i) + \sum_{i \in N_j} y_i \right) \right]^{\ell_j} \\
&= \left[1 - \frac{1}{\ell_j} \left(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \right) \right]^{\ell_j} \\
&\leq \left(1 - \frac{z_j}{\ell_j} \right)^{\ell_j} .
\end{aligned}$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^\ell$ is concave. Hence,

$$\Pr[C_j \text{ satisfied}]$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^\ell$ is concave. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j}$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^\ell$ is concave. Hence,

$$\begin{aligned}\Pr[C_j \text{ satisfied}] &\geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j} \\ &\geq \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \cdot z_j \ .\end{aligned}$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^\ell$ is concave. Hence,

$$\begin{aligned}\Pr[C_j \text{ satisfied}] &\geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j} \\ &\geq \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \cdot z_j \ .\end{aligned}$$

$f''(z) = -\frac{\ell-1}{\ell} \left[1 - \frac{z}{\ell}\right]^{\ell-2} \leq 0$ for $z \in [0, 1]$. Therefore, f is concave.

$$E[W]$$

$$E[W] = \sum_j w_j \Pr[C_j \text{ is satisfied}]$$

$$\begin{aligned} E[W] &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\ &\geq \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j} \right)^{\ell_j} \right] \end{aligned}$$

$$\begin{aligned} E[W] &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\ &\geq \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j} \right)^{\ell_j} \right] \\ &\geq \left(1 - \frac{1}{e} \right) \text{OPT} . \end{aligned}$$

MAXSAT: The better of two

Theorem 85

Choosing the better of the two solutions given by randomized rounding and coin flipping yields a $\frac{3}{4}$ -approximation.

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$E[\max\{W_1, W_2\}]$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$\begin{aligned} E[\max\{W_1, W_2\}] \\ \geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2] \end{aligned}$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

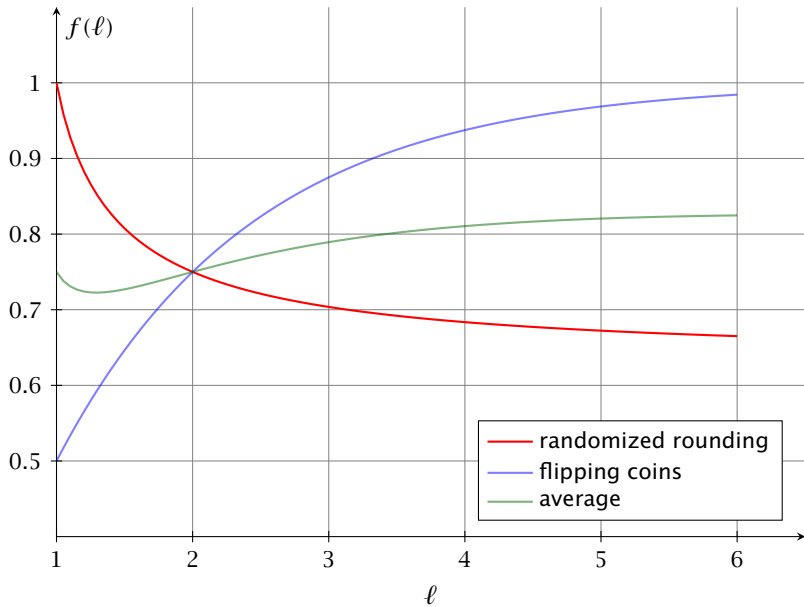
$$\begin{aligned} E[\max\{W_1, W_2\}] &\geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2] \\ &\geq \tfrac{1}{2} \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j} \right] + \tfrac{1}{2} \sum_j w_j \left(1 - \left(\tfrac{1}{2}\right)^{\ell_j}\right) \end{aligned}$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$\begin{aligned} E[\max\{W_1, W_2\}] &\geq E\left[\frac{1}{2}W_1 + \frac{1}{2}W_2\right] \\ &\geq \frac{1}{2} \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j} \right] + \frac{1}{2} \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j} \right) \\ &\geq \sum_j w_j z_j \left[\underbrace{\frac{1}{2} \left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j} \right) + \frac{1}{2} \left(1 - \left(\frac{1}{2}\right)^{\ell_j} \right)}_{\geq \frac{3}{4} \text{ for all integers } \ell_j} \right] \end{aligned}$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$\begin{aligned} & E[\max\{W_1, W_2\}] \\ & \geq E\left[\frac{1}{2}W_1 + \frac{1}{2}W_2\right] \\ & \geq \frac{1}{2} \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j} \right] + \frac{1}{2} \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j} \right) \\ & \geq \sum_j w_j z_j \left[\underbrace{\frac{1}{2} \left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j} \right) + \frac{1}{2} \left(1 - \left(\frac{1}{2}\right)^{\ell_j} \right)}_{\geq \frac{3}{4} \text{ for all integers}} \right] \\ & \geq \frac{3}{4} \text{OPT} \end{aligned}$$



MAXSAT: Nonlinear Randomized Rounding

So far we used **linear** randomized rounding, i.e., the probability that a variable is set to **1**/true was exactly the value of the corresponding variable in the linear program.

We could define a function $f : [0, 1] \rightarrow [0, 1]$ and set x_i to true with probability $f(\gamma_i)$.

MAXSAT: Nonlinear Randomized Rounding

So far we used **linear** randomized rounding, i.e., the probability that a variable is set to 1/true was exactly the value of the corresponding variable in the linear program.

We could define a function $f : [0, 1] \rightarrow [0, 1]$ and set x_i to true with probability $f(y_i)$.

MAXSAT: Nonlinear Randomized Rounding

Let $f : [0, 1] \rightarrow [0, 1]$ be a function with

$$1 - 4^{-x} \leq f(x) \leq 4^{x-1}$$

Theorem 86

Rounding the LP-solution with a function f of the above form gives a $\frac{3}{4}$ -approximation.

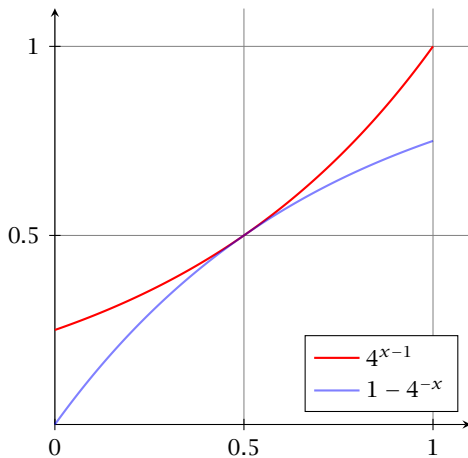
MAXSAT: Nonlinear Randomized Rounding

Let $f : [0, 1] \rightarrow [0, 1]$ be a function with

$$1 - 4^{-x} \leq f(x) \leq 4^{x-1}$$

Theorem 86

Rounding the LP-solution with a function f of the above form gives a $\frac{3}{4}$ -approximation.



$\Pr[C_j \text{ not satisfied}]$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} f(y_i)$$

$$\begin{aligned}\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - f(\gamma_i)) \prod_{i \in N_j} f(\gamma_i) \\ &\leq \prod_{i \in P_j} 4^{-\gamma_i} \prod_{i \in N_j} 4^{\gamma_i - 1}\end{aligned}$$

$$\begin{aligned}\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} f(y_i) \\ &\leq \prod_{i \in P_j} 4^{-y_i} \prod_{i \in N_j} 4^{y_i - 1} \\ &= 4^{-(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i))}\end{aligned}$$

$$\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} f(y_i) \\
&\leq \prod_{i \in P_j} 4^{-y_i} \prod_{i \in N_j} 4^{y_i - 1} \\
&= 4^{-(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i))} \\
&\leq 4^{-z_j}
\end{aligned}$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}]$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j}$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4}z_j \text{ .}$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4}z_j \ .$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4}z_j \text{ .}$$

Therefore,

$$E[W]$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4}z_j \text{ .}$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}]$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4}z_j \ .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}] \geq \frac{3}{4} \sum_j w_j z_j$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4}z_j .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}] \geq \frac{3}{4} \sum_j w_j z_j \geq \frac{3}{4} \text{OPT}$$

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 87 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 87 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 87 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 87 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 87 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Lemma 88

Our ILP-formulation for the MAXSAT problem has integrality gap at most $\frac{3}{4}$.

$$\begin{array}{ll}\max & \sum_j w_j z_j \\ \text{s.t.} & \forall j \quad \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \\ & \forall i \quad y_i \in \{0, 1\} \\ & \forall j \quad z_j \leq 1\end{array}$$

Consider: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- ▶ any solution can satisfy at most 3 clauses
- ▶ we can set $y_1 = y_2 = 1/2$ in the LP; this allows to set $z_1 = z_2 = z_3 = z_4 = 1$
- ▶ hence, the LP has value 4.

Lemma 88

Our ILP-formulation for the MAXSAT problem has integrality gap at most $\frac{3}{4}$.

$$\begin{array}{ll}\max & \sum_j w_j z_j \\ \text{s.t.} & \forall j \quad \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \\ & \forall i \quad y_i \in \{0, 1\} \\ & \forall j \quad z_j \leq 1\end{array}$$

Consider: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- ▶ any solution can satisfy at most 3 clauses
- ▶ we can set $y_1 = y_2 = 1/2$ in the LP; this allows to set $z_1 = z_2 = z_3 = z_4 = 1$
- ▶ hence, the LP has value 4.

MaxCut

Given a weighted graph $G = (V, E, w)$, $w(v) \geq 0$, partition the vertices into two parts. Maximize the weight of edges between the parts.

Trivial 2-approximation

Semidefinite Programming

$$\begin{array}{ll}\max / \min & \sum_{i,j} c_{ij} x_{ij} \\ \text{s.t.} & \forall k \quad \sum_{i,j} a_{ijk} x_{ij} = b_k \\ & \forall i,j \quad x_{ij} = x_{ji} \\ & X = (x_{ij}) \text{ is psd.}\end{array}$$

- ▶ linear objective, linear constraints
- ▶ we can constrain a square matrix of variables to be symmetric positive definite

Vector Programming

$$\begin{array}{ll} \max / \min & \sum_{i,j} c_{ij} (v_i^t v_j) \\ \text{s.t. } \forall k & \sum_{i,j,k} a_{ijk} (v_i^t v_j) = b_k \\ & v_i \in \mathbb{R}^n \end{array}$$

- ▶ variables are vectors in n -dimensional space
- ▶ objective functions and constraints are linear in inner products of the vectors

This is equivalent!

Fact [without proof]

We (essentially) can solve Semidefinite Programs in polynomial time...

Quadratic Programs

Quadratic Program for MaxCut:

$$\begin{array}{ll} \max & \frac{1}{2} \sum_{i,j} w_{ij} (1 - y_i y_j) \\ \forall i & y_i \in \{-1, 1\} \end{array}$$

This is exactly MaxCut!

Semidefinite Relaxation

$$\begin{array}{ll} \max & \frac{1}{2} \sum_{i,j} w_{ij} (1 - v_i^t v_j) \\ & \forall i \quad v_i^t v_i = 1 \\ & \forall i \quad v_i \in \mathbb{R}^n \end{array}$$

- ▶ this is clearly a relaxation
- ▶ the solution will be vectors on the unit sphere

Rounding the SDP-Solution

- ▶ Choose a random vector r such that $r/\|r\|$ is uniformly distributed on the unit sphere.
- ▶ If $r^t v_i > 0$ set $y_i = 1$ else set $y_i = -1$

Rounding the SDP-Solution

Choose the i -th coordinate r_i as a Gaussian with mean 0 and variance 1, i.e., $r_i \sim \mathcal{N}(0, 1)$.

Density function:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{x^2/2}$$

Then

$$\begin{aligned}\Pr[r = (x_1, \dots, x_n)] \\&= \frac{1}{(\sqrt{2\pi})^n} e^{x_1^2/2} \cdot e^{x_2^2/2} \cdot \dots \cdot e^{x_n^2/2} dx_1 \cdot \dots \cdot dx_n \\&= \frac{1}{(\sqrt{2\pi})^n} e^{\frac{1}{2}(x_1^2 + \dots + x_n^2)} dx_1 \cdot \dots \cdot dx_n\end{aligned}$$

Hence the probability for a point only depends on its distance to the origin.

Rounding the SDP-Solution

Choose the i -th coordinate r_i as a Gaussian with mean 0 and variance 1, i.e., $r_i \sim \mathcal{N}(0, 1)$.

Density function:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{x^2/2}$$

Then

$$\begin{aligned}\Pr[r = (x_1, \dots, x_n)] \\&= \frac{1}{(\sqrt{2\pi})^n} e^{x_1^2/2} \cdot e^{x_2^2/2} \cdot \dots \cdot e^{x_n^2/2} dx_1 \cdot \dots \cdot dx_n \\&= \frac{1}{(\sqrt{2\pi})^n} e^{\frac{1}{2}(x_1^2 + \dots + x_n^2)} dx_1 \cdot \dots \cdot dx_n\end{aligned}$$

Hence the probability for a point only depends on its distance to the origin.

Rounding the SDP-Solution

Fact

The projection of r onto two unit vectors e_1 and e_2 are independent and are normally distributed with mean 0 and variance 1 iff e_1 and e_2 are orthogonal.

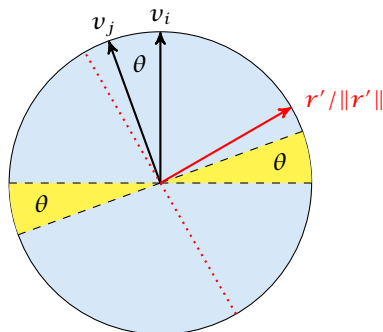
Note that this is clear if e_1 and e_2 are standard basis vectors.

Rounding the SDP-Solution

Corollary

If we project r onto a hyperplane its normalized projection $(r' / \|r'\|)$ is uniformly distributed on the unit circle within the hyperplane.

Rounding the SDP-Solution



- ▶ if the normalized projection falls into the shaded region, v_i and v_j are rounded to different values
- ▶ this happens with probability θ/π

Rounding the SDP-Solution

- ▶ contribution of edge (i, j) to the SDP-relaxation:

$$\frac{1}{2}w_{ij}(1 - v_i^t v_j)$$

- ▶ (expected) contribution of edge (i, j) to the rounded instance $w_{ij} \arccos(v_i^t v_j) / \pi$
- ▶ ratio is at most

$$\min_{x \in [-1, 1]} \frac{2 \arccos(x)}{\pi(1 - x)} \geq 0.878$$

Rounding the SDP-Solution

- ▶ contribution of edge (i, j) to the SDP-relaxation:

$$\frac{1}{2}w_{ij}(1 - v_i^t v_j)$$

- ▶ (expected) contribution of edge (i, j) to the rounded instance $w_{ij} \arccos(v_i^t v_j) / \pi$
- ▶ ratio is at most

$$\min_{x \in [-1, 1]} \frac{2 \arccos(x)}{\pi(1-x)} \geq 0.878$$

Rounding the SDP-Solution

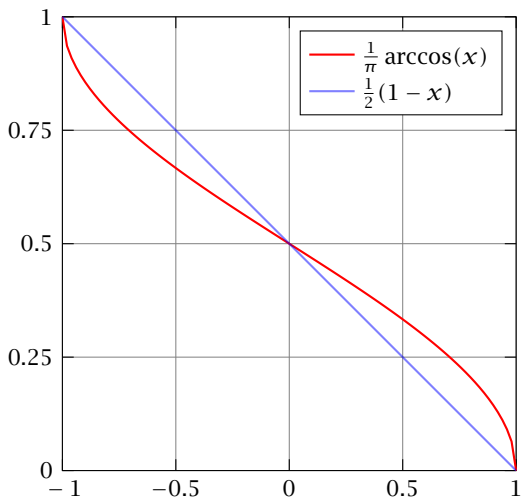
- ▶ contribution of edge (i, j) to the SDP-relaxation:

$$\frac{1}{2}w_{ij}(1 - v_i^t v_j)$$

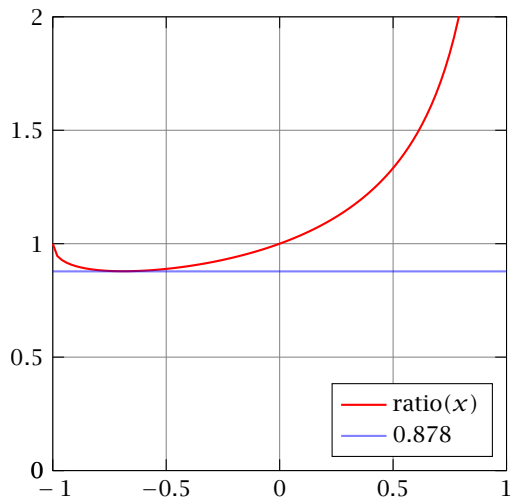
- ▶ (expected) contribution of edge (i, j) to the rounded instance $w_{ij} \arccos(v_i^t v_j) / \pi$
- ▶ ratio is at most

$$\min_{x \in [-1, 1]} \frac{2 \arccos(x)}{\pi(1 - x)} \geq 0.878$$

Rounding the SDP-Solution



Rounding the SDP-Solution



Rounding the SDP-Solution

Theorem 89

Given the unique games conjecture, there is no α -approximation for the maximum cut problem with constant

$$\alpha > \min_{x \in [-1,1]} \frac{2 \arccos(x)}{\pi(1-x)}$$

unless $P = NP$.

Repetition: Primal Dual for Set Cover

Primal Relaxation:

$$\begin{array}{ll}\min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i: u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \geq 0\end{array}$$

Dual Formulation:

$$\begin{array}{ll}\max & \sum_{u \in U} y_u \\ \text{s.t.} & \forall i \in \{1, \dots, k\} \quad \sum_{u: u \in S_i} y_u \leq w_i \\ & y_u \geq 0\end{array}$$

Repetition: Primal Dual for Set Cover

Primal Relaxation:

$$\begin{array}{ll}\min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i: u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \geq 0\end{array}$$

Dual Formulation:

$$\begin{array}{ll}\max & \sum_{u \in U} y_u \\ \text{s.t.} & \forall i \in \{1, \dots, k\} \quad \sum_{u: u \in S_i} y_u \leq w_i \\ & y_u \geq 0\end{array}$$

Repetition: Primal Dual for Set Cover

Algorithm:

- ▶ Start with $y = 0$ (feasible dual solution).
Start with $x = 0$ (integral primal solution that may be infeasible).
- ▶ While x not feasible

Identify an element e that is not covered by current primal integral solution

Increase dual variables y for all sets containing e until e is covered by the dual solution

Repeat until the dual solution is feasible and the primal solution is integral

Repetition: Primal Dual for Set Cover

Algorithm:

- ▶ Start with $y = 0$ (feasible dual solution).
Start with $x = 0$ (integral primal solution that may be infeasible).
- ▶ While x not feasible
 - ▶ Identify an element e that is not covered in current primal integral solution.
 - ▶ Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
 - ▶ If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- ▶ Start with $y = 0$ (feasible dual solution).
Start with $x = 0$ (integral primal solution that may be infeasible).
- ▶ While x not feasible
 - ▶ Identify an element e that is not covered in current primal integral solution.
 - ▶ Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
 - ▶ If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- ▶ Start with $y = 0$ (feasible dual solution).
Start with $x = 0$ (integral primal solution that may be infeasible).
- ▶ While x not feasible
 - ▶ Identify an element e that is not covered in current primal integral solution.
 - ▶ Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
 - ▶ If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- ▶ Start with $y = 0$ (feasible dual solution).
Start with $x = 0$ (integral primal solution that may be infeasible).
- ▶ While x not feasible
 - ▶ Identify an element e that is not covered in current primal integral solution.
 - ▶ Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
 - ▶ If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Analysis:

Repetition: Primal Dual for Set Cover

Analysis:

- ▶ For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

Repetition: Primal Dual for Set Cover

Analysis:

- ▶ For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- ▶ Hence our cost is

Repetition: Primal Dual for Set Cover

Analysis:

- ▶ For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- ▶ Hence our cost is

$$\sum_j w_j x_j$$

Repetition: Primal Dual for Set Cover

Analysis:

- ▶ For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- ▶ Hence our cost is

$$\sum_j w_j x_j = \sum_j \sum_{e \in S_j} y_e$$

Repetition: Primal Dual for Set Cover

Analysis:

- ▶ For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- ▶ Hence our cost is

$$\sum_j w_j x_j = \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e$$

Repetition: Primal Dual for Set Cover

Analysis:

- ▶ For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- ▶ Hence our cost is

$$\begin{aligned} \sum_j w_j x_j &= \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e \\ &\leq f \cdot \sum_e y_e \leq f \cdot \text{OPT} \end{aligned}$$

Note that the constructed pair of primal and dual solution fulfills
primal slackness conditions.

Note that the constructed pair of primal and dual solution fulfills **primal slackness conditions**.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

Note that the constructed pair of primal and dual solution fulfills **primal slackness conditions**.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

If we would also fulfill **dual slackness conditions**

$$y_e > 0 \Rightarrow \sum_{j: e \in S_j} x_j = 1$$

then the solution would be **optimal!!!**

We don't fulfill these constraint but we fulfill an approximate version:

We don't fulfill these constraint but we fulfill an approximate version:

$$y_e > 0 \Rightarrow 1 \leq \sum_{j:e \in S_j} x_j \leq f$$

We don't fulfill these constraint but we fulfill an approximate version:

$$y_e > 0 \Rightarrow 1 \leq \sum_{j:e \in S_j} x_j \leq f$$

This is sufficient to show that the solution is an f -approximation.

Suppose we have a primal/dual pair

$$\begin{array}{ll}\min & \sum_j c_j x_j \\ \text{s.t.} & \forall i \quad \sum_j a_{ij} x_j \geq b_i \\ & \forall j \quad x_j \geq 0\end{array}$$

$$\begin{array}{ll}\max & \sum_i b_i y_i \\ \text{s.t.} & \forall j \quad \sum_i a_{ij} y_i \leq c_j \\ & \forall i \quad y_i \geq 0\end{array}$$

Suppose we have a primal/dual pair

$$\begin{array}{ll}\min & \sum_j c_j x_j \\ \text{s.t.} & \forall i \quad \sum_j a_{ij} x_j \geq b_i \\ & \forall j \quad x_j \geq 0\end{array}$$

$$\begin{array}{ll}\max & \sum_i b_i y_i \\ \text{s.t.} & \forall j \quad \sum_i a_{ij} y_i \leq c_j \\ & \forall i \quad y_i \geq 0\end{array}$$

and solutions that fulfill approximate slackness conditions:

$$x_j > 0 \Rightarrow \sum_i a_{ij} y_i \geq \frac{1}{\alpha} c_j$$

$$y_i > 0 \Rightarrow \sum_j a_{ij} x_j \leq \beta b_i$$

Then

$$\sum_j c_j x_j$$

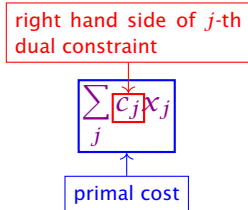
Then

$$\sum_j c_j x_j$$

↑

primal cost

Then



Then

$$\boxed{\sum_j c_j x_j} \leq \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j$$

↑
primal cost

Then

$$\boxed{\sum_j c_j x_j} \leq \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j$$

↑

$$\boxed{\text{primal cost}} = \alpha \sum_i \left(\sum_j a_{ij} x_j \right) y_i$$

Then

$$\begin{aligned} \boxed{\sum_j c_j x_j} &\leq \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j \\ \uparrow \\ \boxed{\text{primal cost}} &= \alpha \sum_i \left(\sum_j a_{ij} x_j \right) y_i \\ &\leq \alpha \beta \cdot \sum_i b_i y_i \end{aligned}$$

Then

$$\begin{aligned} \boxed{\sum_j c_j x_j} &\leq \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j \\ \text{primal cost} &= \alpha \sum_i \left(\sum_j a_{ij} x_j \right) y_i \\ &\leq \alpha \beta \cdot \boxed{\sum_i b_i y_i} \\ &\quad \uparrow \\ &\quad \text{dual objective} \end{aligned}$$

Feedback Vertex Set for Undirected Graphs

- ▶ Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.

Feedback Vertex Set for Undirected Graphs

- ▶ Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.
- ▶ Choose a minimum cost subset of vertices s.t. every cycle contains at least one vertex.

We can encode this as an instance of Set Cover

- ▶ Each vertex can be viewed as a set that contains some cycles.

We can encode this as an instance of Set Cover

- ▶ Each vertex can be viewed as a set that contains some cycles.
- ▶ However, this encoding gives a Set Cover instance of non-polynomial size.

We can encode this as an instance of Set Cover

- ▶ Each vertex can be viewed as a set that contains some cycles.
- ▶ However, this encoding gives a Set Cover instance of non-polynomial size.
- ▶ The $O(\log n)$ -approximation for Set Cover does not help us to get a good solution.

Let $\mathcal{C}$ denote the set of all cycles (where a cycle is identified by its set of vertices)

Let $\mathcal{C}$ denote the set of all cycles (where a cycle is identified by its set of vertices)

Primal Relaxation:

$$\begin{array}{ll} \min & \sum_v w_v x_v \\ \text{s.t.} & \forall C \in \mathcal{C} \quad \sum_{v \in C} x_v \geq 1 \\ & \forall v \quad x_v \geq 0 \end{array}$$

Dual Formulation:

$$\begin{array}{ll} \max & \sum_{C \in \mathcal{C}} y_C \\ \text{s.t.} & \forall v \in V \quad \sum_{C: v \in C} y_C \leq w_v \\ & \forall C \quad y_C \geq 0 \end{array}$$

If we perform the previous dual technique for Set Cover we get the following:

- ▶ Start with $x = 0$ and $y = 0$

If we perform the previous dual technique for Set Cover we get the following:

- ▶ Start with $x = 0$ and $y = 0$
- ▶ While there is a cycle C that is not covered (does not contain a chosen vertex).

If we perform the previous dual technique for Set Cover we get the following:

- ▶ Start with $x = 0$ and $y = 0$
- ▶ While there is a cycle C that is not covered (does not contain a chosen vertex).
 - ▶ Increase y_C until dual constraint for some vertex v becomes tight.

If we perform the previous dual technique for Set Cover we get the following:

- ▶ Start with $x = 0$ and $y = 0$
- ▶ While there is a cycle C that is not covered (does not contain a chosen vertex).
 - ▶ Increase y_C until dual constraint for some vertex v becomes tight.
 - ▶ set $x_v = 1$.

Then

$$\sum_v w_v x_v$$

Then

$$\sum_v w_v x_v = \sum_v \sum_{C: v \in C} y_C x_v$$

Then

$$\begin{aligned}\sum_v w_v x_v &= \sum_v \sum_{C:v \in C} y_C x_v \\ &= \sum_{v \in S} \sum_{C:v \in C} y_C\end{aligned}$$

where S is the set of vertices we choose.

Then

$$\begin{aligned}\sum_v w_v x_v &= \sum_v \sum_{C:v \in C} y_C x_v \\ &= \sum_{v \in S} \sum_{C:v \in C} y_C \\ &= \sum_C |S \cap C| \cdot y_C\end{aligned}$$

where S is the set of vertices we choose.

Then

$$\begin{aligned}\sum_v w_v x_v &= \sum_v \sum_{C: v \in C} y_C x_v \\ &= \sum_{v \in S} \sum_{C: v \in C} y_C \\ &= \sum_C |S \cap C| \cdot y_C\end{aligned}$$

where S is the set of vertices we choose.

If every cycle is short we get a good approximation ratio, but this is unrealistic.

Algorithm 1 FeedbackVertexSet

```
1:  $\gamma \leftarrow 0$ 
2:  $x \leftarrow 0$ 
3: while exists cycle  $C$  in  $G$  do
4:   increase  $\gamma_C$  until there is  $v \in C$  s.t.  $\sum_{C:v \in C} \gamma_C = w_v$ 
5:    $x_v = 1$ 
6:   remove  $v$  from  $G$ 
7:   repeatedly remove vertices of degree 1 from  $G$ 
```

Idea:

Always choose a short cycle that is not covered. If we always find a cycle of length at most α we get an α -approximation.

Idea:

Always choose a short cycle that is not covered. If we always find a cycle of length at most α we get an α -approximation.

Observation:

For any path P of vertices of degree 2 in G the algorithm chooses at most one vertex from P .

Observation:

If we always choose a cycle for which the number of vertices of degree at least 3 is at most α we get a 2α -approximation.

Observation:

If we always choose a cycle for which the number of vertices of degree at least 3 is at most α we get a 2α -approximation.

Theorem 90

In any graph with no vertices of degree 1, there always exists a cycle that has at most $\mathcal{O}(\log n)$ vertices of degree 3 or more. We can find such a cycle in linear time.

This means we have

$$y_C > 0 \Rightarrow |S \cap C| \leq \mathcal{O}(\log n) .$$

Primal Dual for Shortest Path

Given a graph $G = (V, E)$ with two nodes $s, t \in V$ and edge-weights $c : E \rightarrow \mathbb{R}^+$ find a shortest path between s and t w.r.t. edge-weights c .

$$\begin{array}{ll} \min & \sum_e c(e) x_e \\ \text{s.t.} & \forall S \in \mathcal{S} \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S , and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

Given a graph $G = (V, E)$ with two nodes $s, t \in V$ and edge-weights $c : E \rightarrow \mathbb{R}^+$ find a shortest path between s and t w.r.t. edge-weights c .

$$\begin{array}{ll} \min & \sum_e c(e) x_e \\ \text{s.t.} & \forall S \in \mathcal{S} \quad \sum_{e: \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S , and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

The Dual:

$$\begin{array}{ll}\max & \sum_S y_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S: e \in \delta(S)} y_S \leq c(e) \\ & \forall S \in \mathcal{S} \quad y_S \geq 0\end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S , and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

The Dual:

$$\begin{array}{ll}\max & \sum_S y_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S: e \in \delta(S)} y_S \leq c(e) \\ & \forall S \in \mathcal{S} \quad y_S \geq 0\end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S , and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surrounding the set S .

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surrounding the set S .

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surrounding the set S .

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surrounding the set S .

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Algorithm 1 PrimalDualShortestPath

- 1: $\gamma \leftarrow 0$
- 2: $F \leftarrow \emptyset$
- 3: **while** there is no s - t path in (V, F) **do**
- 4: Let C be the connected component of (V, F) containing s
- 5: Increase γ_C until there is an edge $e' \in \delta(C)$ such that $\sum_{S: e' \in \delta(S)} \gamma_S = c(e')$.
- 6: $F \leftarrow F \cup \{e'\}$
- 7: **Let** P **be an** s - t **path in** (V, F)
- 8: **return** P

Lemma 91

At each point in time the set F forms a tree.

Proof:

Initially F contains only the current selected component C .
If F is not a tree, then it contains a cycle. This component C and add
some edges from C to C .
Since C is a tree, it contains at most one end-point of the new edge. In the
edge cannot close a cycle.

Lemma 91

At each point in time the set F forms a tree.

Proof:

- ▶ In each iteration we take the current connected component from (V, F) that contains s (call this component C) and add some edge from $\delta(C)$ to F .
- ▶ Since, at most one end-point of the new edge is in C the edge cannot close a cycle.

Lemma 91

At each point in time the set F forms a tree.

Proof:

- ▶ In each iteration we take the current connected component from (V, F) that contains s (call this component C) and add some edge from $\delta(C)$ to F .
- ▶ Since, at most one end-point of the new edge is in C the edge cannot close a cycle.

$$\sum_{e \in P} c(e)$$

$$\sum_{e \in P} c(e) = \sum_{e \in P} \sum_{S: e \in \delta(S)} y_S$$

$$\begin{aligned}
 \sum_{e \in P} c(e) &= \sum_{e \in P} \sum_{S: e \in \delta(S)} y_S \\
 &= \sum_{S: s \in S, t \notin S} |P \cap \delta(S)| \cdot y_S .
 \end{aligned}$$

$$\begin{aligned}
 \sum_{e \in P} c(e) &= \sum_{e \in P} \sum_{S: e \in \delta(S)} y_S \\
 &= \sum_{S: s \in S, t \notin S} |P \cap \delta(S)| \cdot y_S .
 \end{aligned}$$

If we can show that $y_S > 0$ implies $|P \cap \delta(S)| = 1$ gives

$$\sum_{e \in P} c(e) = \sum_S y_S \leq \text{OPT}$$

by weak duality.

$$\begin{aligned}
 \sum_{e \in P} c(e) &= \sum_{e \in P} \sum_{S: e \in \delta(S)} y_S \\
 &= \sum_{S: s \in S, t \notin S} |P \cap \delta(S)| \cdot y_S .
 \end{aligned}$$

If we can show that $y_S > 0$ implies $|P \cap \delta(S)| = 1$ gives

$$\sum_{e \in P} c(e) = \sum_S y_S \leq \text{OPT}$$

by weak duality.

Hence, we find a shortest path.

If $\delta(S)$ contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S , S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $\delta(S)$ contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S , S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $\delta(S)$ contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S , S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $\delta(S)$ contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S , S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $\delta(S)$ contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S , S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

Steiner Forest Problem:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i , $i = 1, \dots, k$, and a cost function $c : E \rightarrow \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F .

$$\begin{array}{ll} \min & \sum_e c(e)x_e \\ \text{s.t.} & \forall S \subseteq V : S \in S_i \text{ for some } i \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here S_i contains all sets S such that $s_i \in S$ and $t_i \notin S$.

Steiner Forest Problem:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i , $i = 1, \dots, k$, and a cost function $c : E \rightarrow \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F .

$$\begin{array}{ll} \min & \sum_e c(e)x_e \\ \text{s.t.} & \forall S \subseteq V : S \in \mathcal{S}_i \text{ for some } i \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here $\mathcal{S}_i$ contains all sets S such that $s_i \in S$ and $t_i \notin S$.

Steiner Forest Problem:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i , $i = 1, \dots, k$, and a cost function $c : E \rightarrow \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F .

$$\begin{array}{ll} \min & \sum_e c(e) x_e \\ \text{s.t.} & \forall S \subseteq V : S \in S_i \text{ for some } i \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here S_i contains all sets S such that $s_i \in S$ and $t_i \notin S$.

$$\begin{array}{ll}
 \max & \sum_{S: \exists i \text{ s.t. } S \in s_i} \mathcal{Y}_S \\
 \text{s.t.} & \forall e \in E \quad \sum_{S: e \in \delta(S)} \mathcal{Y}_S \leq c(e) \\
 & \mathcal{Y}_S \geq 0
 \end{array}$$

The difference to the dual of the shortest path problem is that we have many more variables (sets for which we can generate a moat of non-zero width).

Algorithm 1 FirstTry

```
1:  $\gamma \leftarrow 0$ 
2:  $F \leftarrow \emptyset$ 
3: while not all  $s_i$ - $t_i$  pairs connected in  $F$  do
4:   Let  $C$  be some connected component of  $(V, F)$  such
     that  $|C \cap \{s_i, t_i\}| = 1$  for some  $i$ .
5:   Increase  $\gamma_C$  until there is an edge  $e' \in \delta(C)$  s.t.
      $\sum_{S \in S_i: e' \in \delta(S)} \gamma_S = c_{e'}$ 
6:    $F \leftarrow F \cup \{e'\}$ 
7: return  $\bigcup_i P_i$ 
```

$$\sum_{e \in F} c(e)$$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S$$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- ▶ Take a complete graph on $k + 1$ vertices $v_0, v_1, \dots, v_k$.

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- ▶ Take a complete graph on $k + 1$ vertices $v_0, v_1, \dots, v_k$.
- ▶ The i -th pair is $v_0 - v_i$.

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- ▶ Take a complete graph on $k + 1$ vertices $v_0, v_1, \dots, v_k$.
- ▶ The i -th pair is $v_0 - v_i$.
- ▶ The first component C could be $\{v_0\}$.

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- ▶ Take a complete graph on $k + 1$ vertices $v_0, v_1, \dots, v_k$.
- ▶ The i -th pair is $v_0 - v_i$.
- ▶ The first component C could be $\{v_0\}$.
- ▶ We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S.$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- ▶ Take a complete graph on $k + 1$ vertices $v_0, v_1, \dots, v_k$.
- ▶ The i -th pair is $v_0 - v_i$.
- ▶ The first component C could be $\{v_0\}$.
- ▶ We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.
- ▶ The final set F contains all edges $\{v_0, v_i\}$, $i = 1, \dots, k$.

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

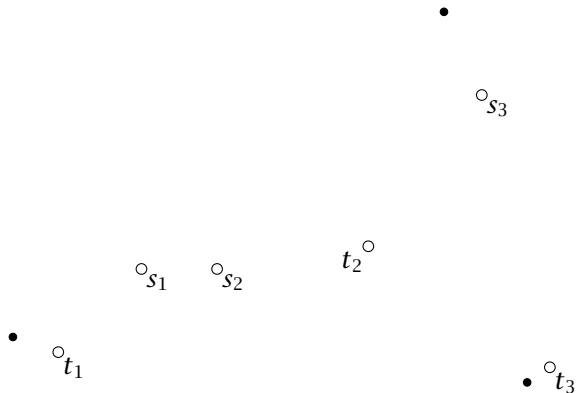
- ▶ Take a complete graph on $k + 1$ vertices $v_0, v_1, \dots, v_k$.
- ▶ The i -th pair is $v_0 - v_i$.
- ▶ The first component C could be $\{v_0\}$.
- ▶ We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.
- ▶ The final set F contains all edges $\{v_0, v_i\}$, $i = 1, \dots, k$.
- ▶ $y_{\{v_0\}} > 0$ but $|\delta(\{v_0\}) \cap F| = k$.

Algorithm 1 SecondTry

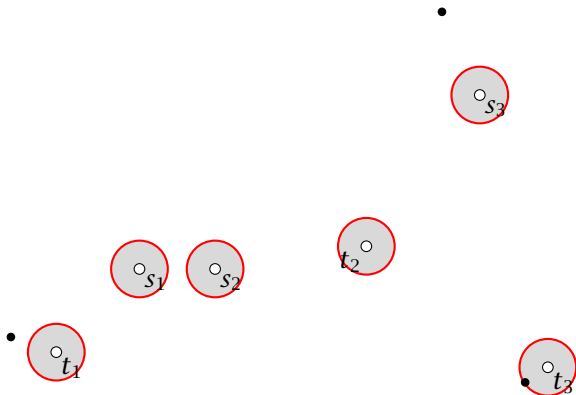
```
1:  $\gamma \leftarrow 0; F \leftarrow \emptyset; \ell \leftarrow 0$ 
2: while not all  $s_i$ - $t_i$  pairs connected in  $F$  do
3:    $\ell \leftarrow \ell + 1$ 
4:   Let  $\mathfrak{C}$  be set of all connected components  $C$  of  $(V, F)$ 
     such that  $|C \cap \{s_i, t_i\}| = 1$  for some  $i$ .
5:   Increase  $\gamma_C$  for all  $C \in \mathfrak{C}$  uniformly until for some edge
      $e_\ell \in \delta(C')$ ,  $C' \in \mathfrak{C}$  s.t.  $\sum_{S: e_\ell \in \delta(S)} \gamma_S = c_{e_\ell}$ 
6:    $F \leftarrow F \cup \{e_\ell\}$ 
7:  $F' \leftarrow F$ 
8: for  $k \leftarrow \ell$  downto 1 do // reverse deletion
9:   if  $F' - e_k$  is feasible solution then
10:     remove  $e_k$  from  $F'$ 
11: return  $F'$ 
```

The reverse deletion step is not strictly necessary this way. It would also be sufficient to simply delete all unnecessary edges in any order.

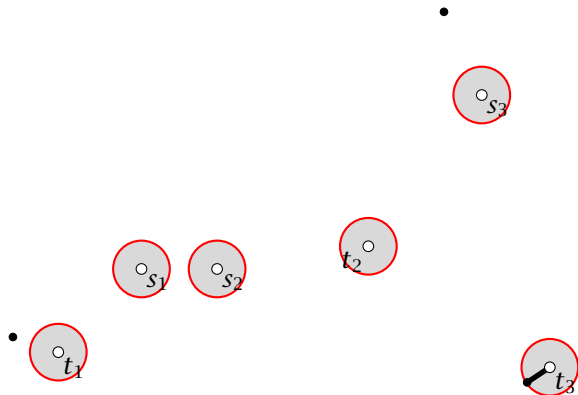
Example



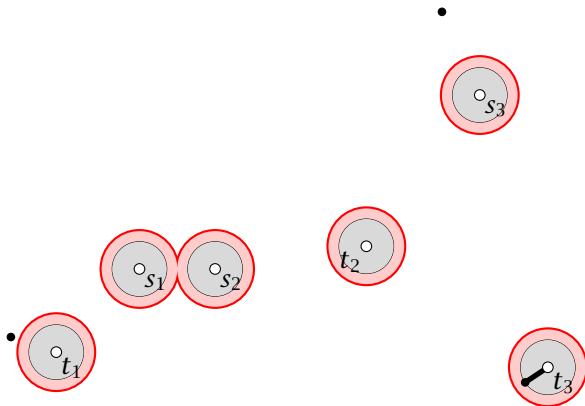
Example



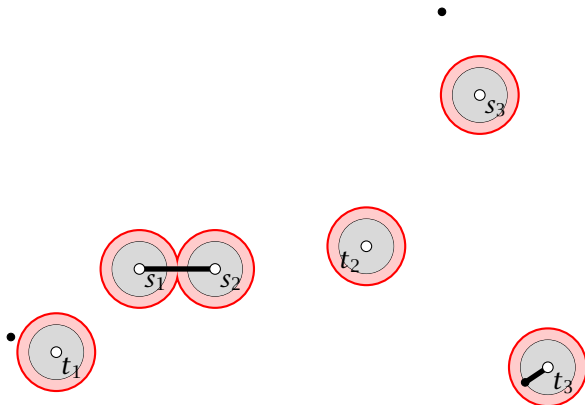
Example



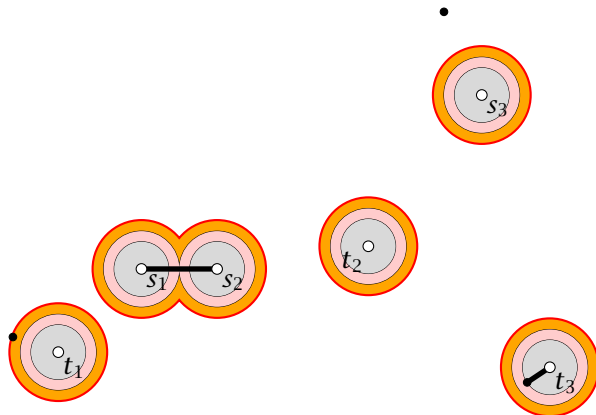
Example



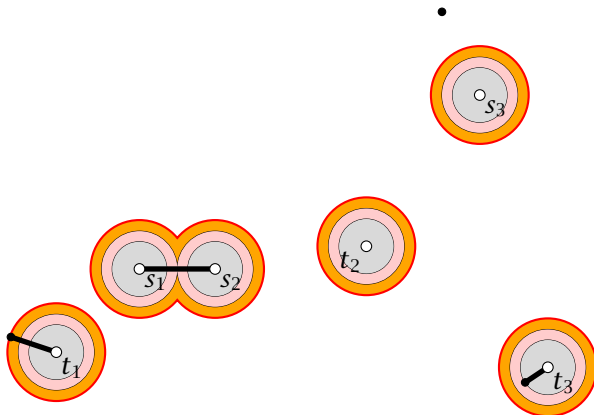
Example



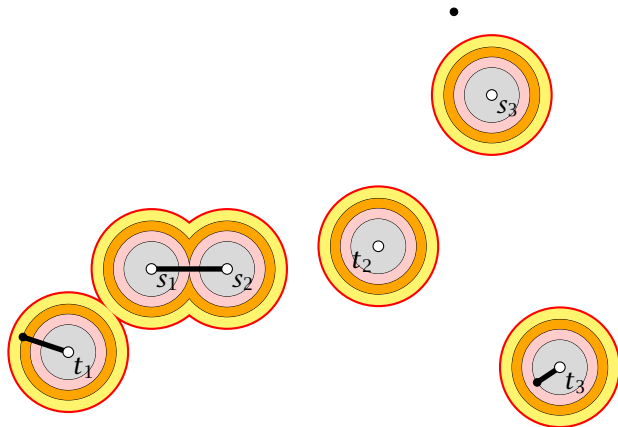
Example



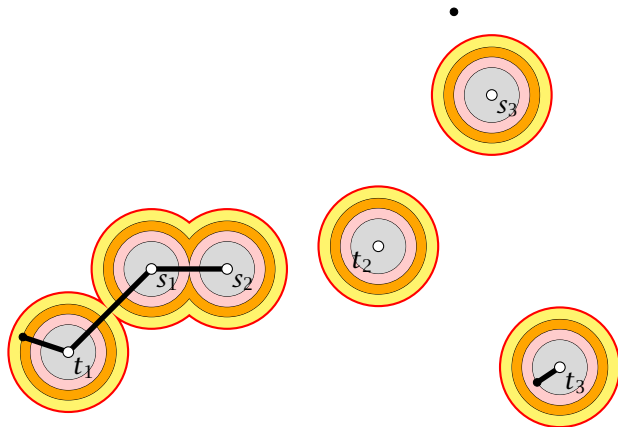
Example



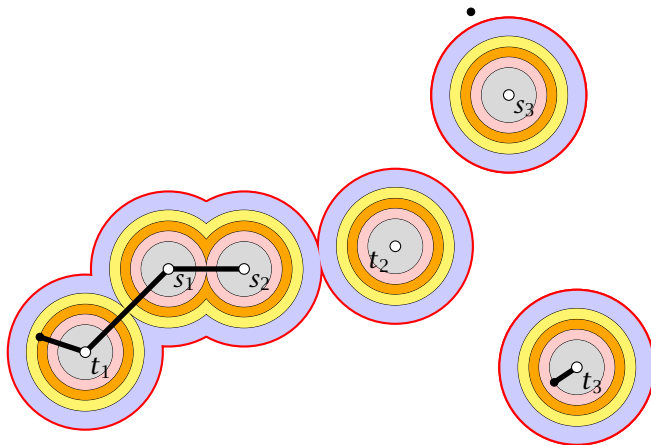
Example



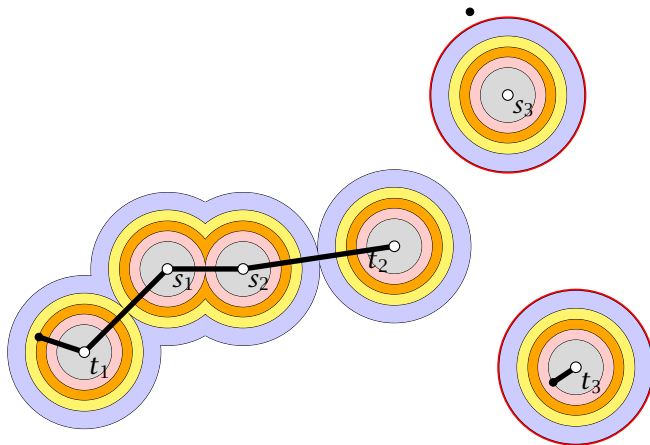
Example



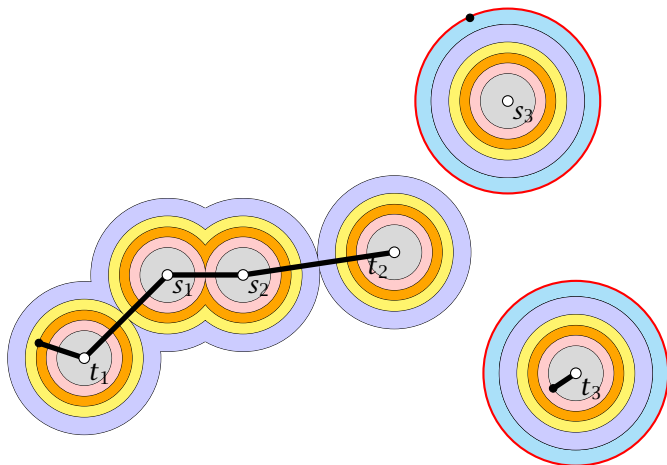
Example



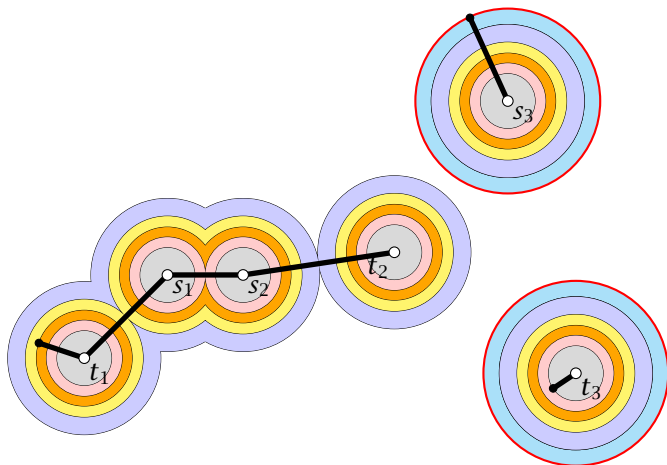
Example



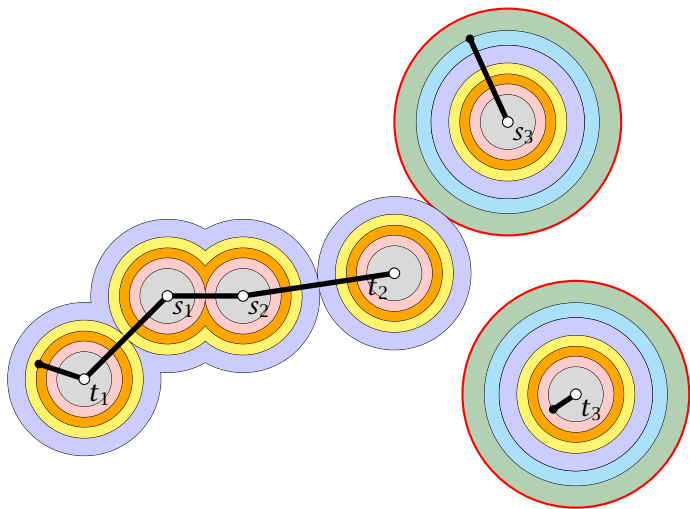
Example



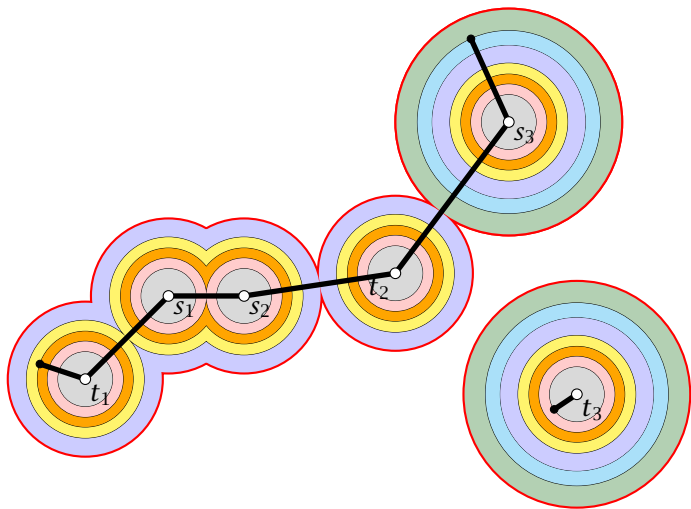
Example



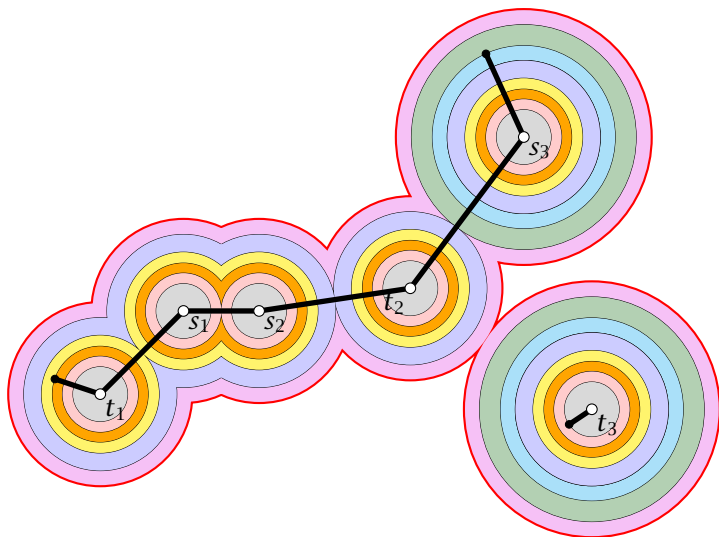
Example



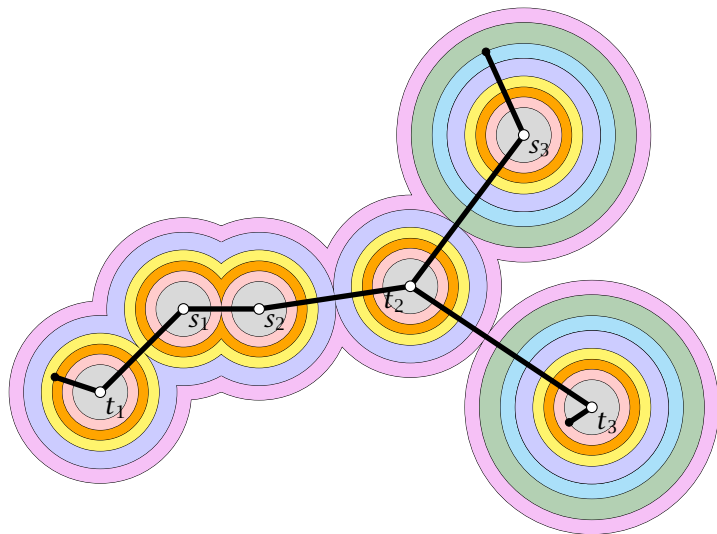
Example



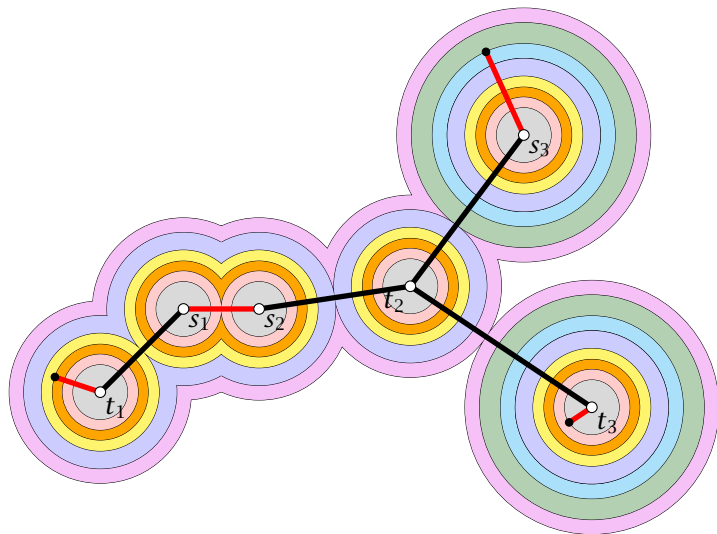
Example



Example



Example



Lemma 92

For any $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

This means that the number of times a moat from $\mathcal{C}$ is crossed in the final solution is at most twice the number of moats.

Proof: later...

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} \gamma_S = \sum_S |F' \cap \delta(S)| \cdot \gamma_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot \gamma_S \leq 2 \sum_S \gamma_S$$

in the i -th iteration the increase of the left-hand side is

$$2 \sum_{S \in \mathcal{S}_i} \gamma_S$$

and the increase of the right hand side is $2 \sum_{S \in \mathcal{S}_i} \gamma_S$.

Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} \gamma_S = \sum_S |F' \cap \delta(S)| \cdot \gamma_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot \gamma_S \leq 2 \sum_S \gamma_S$$

in the i -th iteration the increase of the left-hand side is

$$2 \sum_{S \in \mathcal{S}_i} \gamma_S$$

and the increase of the right hand side is $\sum_{S \in \mathcal{S}_i} \gamma_S$.

Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \leq 2 \sum_S y_S$$

in the i -th iteration the increase of the left-hand side is

$$2 \sum_{S \in \mathcal{S}_i} y_S$$

and the increase of the right hand side is $\sum_{S \in \mathcal{S}_i} y_S$.

Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \leq 2 \sum_S y_S$$

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \leq 2 \sum_S y_S$$

- In the i -th iteration the increase of the left-hand side is

$$\epsilon \sum_{C \in \mathcal{C}} |F' \cap \delta(C)|$$

and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.

- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \leq 2 \sum_S y_S$$

- In the i -th iteration the increase of the left-hand side is

$$\epsilon \sum_{C \in \mathbb{C}} |F' \cap \delta(C)|$$

and the increase of the right hand side is $2\epsilon|\mathbb{C}|$.

- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

Lemma 93

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

At any point during the algorithm, the set of edges forms a forest F' . The set of connected components $\mathcal{C}$ is a partition of V into connected components.

For each component $C \in \mathcal{C}$, the set of edges $\delta(C) \cap F'$ is a forest. The number of edges in $\delta(C) \cap F'$ is at most $|C| - 1$.

Let $\mathcal{C} = \{C_1, \dots, C_k\}$.

The total number of edges in $\delta(\mathcal{C}) \cap F'$ is at most $\sum_{C \in \mathcal{C}} (|C| - 1) = |\mathcal{C}|$.

Lemma 93

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

- ▶ At any point during the algorithm the set of edges forms a forest (why?).
- ▶ Fix iteration i . Let F_i be the set of edges in F at the beginning of the iteration.
- ▶ Let $H = F' - F_i$.
- ▶ All edges in H are necessary for the solution.

Lemma 93

For any set of connected components $\mathbb{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathbb{C}} |\delta(C) \cap F'| \leq 2|\mathbb{C}|$$

Proof:

- ▶ At any point during the algorithm the set of edges forms a forest (why?).
- ▶ Fix iteration i . Let F_i be the set of edges in F at the beginning of the iteration.
- ▶ Let $H = F' - F_i$.
- ▶ All edges in H are necessary for the solution.

Lemma 93

For any set of connected components $\mathbb{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathbb{C}} |\delta(C) \cap F'| \leq 2|\mathbb{C}|$$

Proof:

- ▶ At any point during the algorithm the set of edges forms a forest (why?).
- ▶ Fix iteration i . Let F_i be the set of edges in F at the beginning of the iteration.
- ▶ Let $H = F' - F_i$.
- ▶ All edges in H are necessary for the solution.

Lemma 93

For any set of connected components $\mathbb{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathbb{C}} |\delta(C) \cap F'| \leq 2|\mathbb{C}|$$

Proof:

- ▶ At any point during the algorithm the set of edges forms a forest (why?).
- ▶ Fix iteration i . Let F_i be the set of edges in F at the beginning of the iteration.
- ▶ Let $H = F' - F_i$.
- ▶ All edges in H are necessary for the solution.

- ▶ Contract all edges in F_i into single vertices V' .
- ▶ We can consider the forest H on the set of vertices V' .
- ▶ Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- ▶ Color a vertex $v \in V'$ red if it corresponds to a component from $\mathbb{C}$ (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- ▶ We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in \mathbb{C}} |\delta(C) \cap V'| \stackrel{?}{\leq} 2|\mathbb{C}| = 2|R|$$

- ▶ Contract all edges in F_i into single vertices V' .
- ▶ We can consider the forest H on the set of vertices V' .
- ▶ Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- ▶ Color a vertex $v \in V'$ red if it corresponds to a component from $\mathbb{C}$ (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- ▶ We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in \mathbb{C}} |\delta(C) \cap V'| \stackrel{?}{\leq} 2|\mathbb{C}| = 2|R|$$

- ▶ Contract all edges in F_i into single vertices V' .
- ▶ We can consider the forest H on the set of vertices V' .
- ▶ Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- ▶ Color a vertex $v \in V'$ red if it corresponds to a component from $\mathbb{C}$ (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- ▶ We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in \mathbb{C}} |\delta(C) \cap V'| \stackrel{?}{\leq} 2|\mathbb{C}| = 2|R|$$

- ▶ Contract all edges in F_i into single vertices V' .
- ▶ We can consider the forest H on the set of vertices V' .
- ▶ Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- ▶ Color a vertex $v \in V'$ red if it corresponds to a component from $\mathbb{C}$ (an active component). Otherwise, color it blue. (Let B be the set of blue vertices (with non-zero degree) and R the set of red vertices)
- ▶ We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in \mathbb{C}} |\delta(C) \cap V'| \stackrel{?}{\leq} 2|\mathbb{C}| = 2|R|$$

- ▶ Contract all edges in F_i into single vertices V' .
- ▶ We can consider the forest H on the set of vertices V' .
- ▶ Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- ▶ Color a vertex $v \in V'$ red if it corresponds to a component from $\mathbb{C}$ (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- ▶ We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in \mathbb{C}} |\delta(C) \cap V'| \stackrel{?}{\leq} 2|\mathbb{C}| = 2|R|$$

- ▶ Suppose that no node in B has degree one.

- ▶ Suppose that no node in B has degree one.
- ▶ Then

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\sum_{v \in R} \deg(v)$$

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v)$$

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B|\end{aligned}$$

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- ▶ Every blue vertex with non-zero degree must have degree at least two.

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- ▶ Every blue vertex with non-zero degree must have degree at least two.
 - ▶ Suppose not. The single edge connecting $b \in B$ comes from H , and, hence, is necessary.

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- ▶ Every blue vertex with non-zero degree must have degree at least two.
 - ▶ Suppose not. The single edge connecting $b \in B$ comes from H , and, hence, is necessary.
 - ▶ But this means that the cluster corresponding to b must separate a source-target pair.

- ▶ Suppose that no node in B has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- ▶ Every blue vertex with non-zero degree must have degree at least two.
 - ▶ Suppose not. The single edge connecting $b \in B$ comes from H , and, hence, is necessary.
 - ▶ But this means that the cluster corresponding to b must separate a source-target pair.
 - ▶ But then it must be a red node.

17 Cuts & Metrics

Shortest Path

$$\begin{array}{ll}\min & \sum_e c(e) x_e \\ \text{s.t.} & \forall S \in \mathcal{S} \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\}\end{array}$$

$\mathcal{S}$ is the set of subsets that separate s from t .

The Dual:

$$\begin{array}{ll}\max & \sum_S \gamma_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S: e \in \delta(S)} \gamma_S \leq c(e) \\ & \forall S \in \mathcal{S} \quad \gamma_S \geq 0\end{array}$$

The **Separation Problem** for the Shortest Path LP is the Minimum Cut Problem.

17 Cuts & Metrics

Shortest Path

$$\begin{array}{ll}\min & \sum_e c(e) x_e \\ \text{s.t.} & \forall S \in \mathcal{S} \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \geq 0\end{array}$$

$\mathcal{S}$ is the set of subsets that separate s from t .

The Dual:

$$\begin{array}{ll}\max & \sum_S y_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S: e \in \delta(S)} y_S \leq c(e) \\ & \forall S \in \mathcal{S} \quad y_S \geq 0\end{array}$$

The **Separation Problem** for the Shortest Path LP is the Minimum Cut Problem.

17 Cuts & Metrics

Shortest Path

$$\begin{array}{ll}\min & \sum_e c(e)x_e \\ \text{s.t.} & \forall S \in \mathcal{S} \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \geq 0\end{array}$$

$\mathcal{S}$ is the set of subsets that separate s from t .

The Dual:

$$\begin{array}{ll}\max & \sum_S y_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S: e \in \delta(S)} y_S \leq c(e) \\ & \forall S \in \mathcal{S} \quad y_S \geq 0\end{array}$$

The **Separation Problem** for the Shortest Path LP is the Minimum Cut Problem.

17 Cuts & Metrics

Minimum Cut

$$\begin{array}{ll}\min & \sum_e c(e)x_e \\ \text{s.t.} & \forall P \in \mathcal{P} \quad \sum_{e \in P} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0,1\}\end{array}$$

$\mathcal{P}$ is the set of path that connect s and t .

The Dual:

$$\begin{array}{ll}\max & \sum_P y_P \\ \text{s.t.} & \forall e \in E \quad \sum_{P:e \in P} y_P \leq c(e) \\ & \forall P \in \mathcal{P} \quad y_P \geq 0\end{array}$$

The **Separation Problem** for the Minimum Cut LP is the Shortest Path Problem.

17 Cuts & Metrics

Minimum Cut

$$\begin{array}{ll}\min & \sum_e c(e)x_e \\ \text{s.t.} & \forall P \in \mathcal{P} \quad \sum_{e \in P} x_e \geq 1 \\ & \forall e \in E \quad x_e \geq 0\end{array}$$

$\mathcal{P}$ is the set of path that connect s and t .

The Dual:

$$\begin{array}{ll}\max & \sum_P y_P \\ \text{s.t.} & \forall e \in E \quad \sum_{P: e \in P} y_P \leq c(e) \\ & \forall P \in \mathcal{P} \quad y_P \geq 0\end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

17 Cuts & Metrics

Minimum Cut

$$\begin{array}{ll}\min & \sum_e c(e)x_e \\ \text{s.t.} & \forall P \in \mathcal{P} \quad \sum_{e \in P} x_e \geq 1 \\ & \forall e \in E \quad x_e \geq 0\end{array}$$

$\mathcal{P}$ is the set of path that connect s and t .

The Dual:

$$\begin{array}{ll}\max & \sum_P y_P \\ \text{s.t.} & \forall e \in E \quad \sum_{P: e \in P} y_P \leq c(e) \\ & \forall P \in \mathcal{P} \quad y_P \geq 0\end{array}$$

The **Separation Problem** for the Minimum Cut LP is the Shortest Path Problem.

17 Cuts & Metrics

Minimum Cut

$$\begin{array}{ll} \min & \sum_e c(e) \ell_e \\ \text{s.t.} & \forall P \in \mathcal{P} \quad \sum_{e \in P} \ell_e \geq 1 \\ & \forall e \in E \quad \ell_e \geq 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect s and t .

The Dual:

$$\begin{array}{ll} \max & \sum_P f_P \\ \text{s.t.} & \forall e \in E \quad \sum_{P: e \in P} f_P \leq c(e) \\ & \forall P \in \mathcal{P} \quad f_P \geq 0 \end{array}$$

The **Separation Problem** for the Minimum Cut LP is the Shortest Path Problem.

17 Cuts & Metrics

Observations:

Suppose that ℓ_e -values are solution to Minimum Cut LP.

- ▶ We can view ℓ_e as defining the **length** of an edge.
- ▶ Define $d(u, v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the **Shortest Path Metric** induced by ℓ_e .
- ▶ We have $d(u, v) = \ell_e$ for every edge $e = (u, v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t .

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

17 Cuts & Metrics

Observations:

Suppose that ℓ_e -values are solution to Minimum Cut LP.

- ▶ We can view ℓ_e as defining the **length** of an edge.
- ▶ Define $d(u, v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the **Shortest Path Metric** induced by ℓ_e .
- ▶ We have $d(u, v) = \ell_e$ for every edge $e = (u, v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t .

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

17 Cuts & Metrics

Observations:

Suppose that ℓ_e -values are solution to Minimum Cut LP.

- ▶ We can view ℓ_e as defining the **length** of an edge.
- ▶ Define $d(u, v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the **Shortest Path Metric** induced by ℓ_e .
- ▶ We have $d(u, v) = \ell_e$ for every edge $e = (u, v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t .

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

17 Cuts & Metrics

Observations:

Suppose that ℓ_e -values are solution to Minimum Cut LP.

- ▶ We can view ℓ_e as defining the **length** of an edge.
- ▶ Define $d(u, v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the **Shortest Path Metric** induced by ℓ_e .
- ▶ We have $d(u, v) = \ell_e$ for every edge $e = (u, v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t .

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

How do we round the LP?

- ▶ Let $B(s, r)$ be the ball of radius r around s (w.r.t. metric d).
Formally:

$$B = \{v \in V \mid d(s, v) \leq r\}$$

- ▶ For $0 \leq r < 1$, $B(s, r)$ is an s - t -cut.

Which value of r should we choose? choose randomly!!!

Formally:

choose r u.a.r. (uniformly at random) from interval $[0, 1)$

How do we round the LP?

- ▶ Let $B(s, r)$ be the ball of radius r around s (w.r.t. metric d).
Formally:

$$B = \{v \in V \mid d(s, v) \leq r\}$$

- ▶ For $0 \leq r < 1$, $B(s, r)$ is an s - t -cut.

Which value of r should we choose? choose randomly!!!

Formally:

choose r u.a.r. (uniformly at random) from interval $[0, 1)$

How do we round the LP?

- ▶ Let $B(s, r)$ be the ball of radius r around s (w.r.t. metric d).
Formally:

$$B = \{v \in V \mid d(s, v) \leq r\}$$

- ▶ For $0 \leq r < 1$, $B(s, r)$ is an s - t -cut.

Which value of r should we choose? **choose randomly!!!**

Formally:

choose r u.a.r. (uniformly at random) from interval $[0, 1)$

How do we round the LP?

- ▶ Let $B(s, r)$ be the ball of radius r around s (w.r.t. metric d).
Formally:

$$B = \{v \in V \mid d(s, v) \leq r\}$$

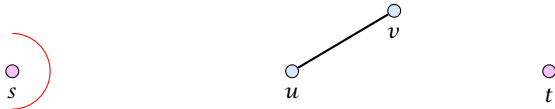
- ▶ For $0 \leq r < 1$, $B(s, r)$ is an s - t -cut.

Which value of r should we choose? **choose randomly!!!**

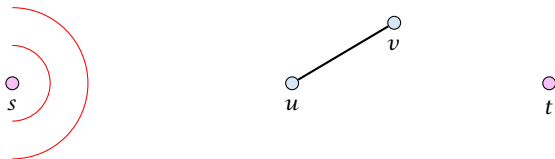
Formally:

choose r **u.a.r.** (**uniformly at random**) from interval $[0, 1)$

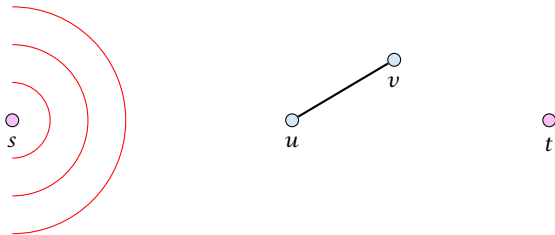
What is the probability that an edge (u, v) is in the cut?



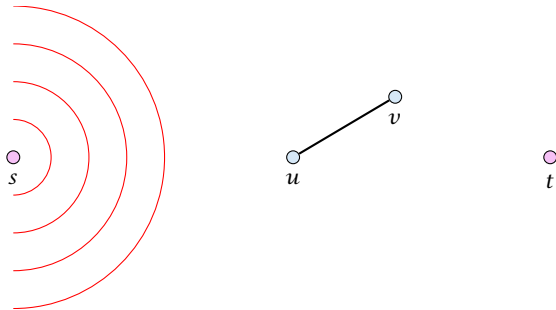
What is the probability that an edge (u, v) is in the cut?



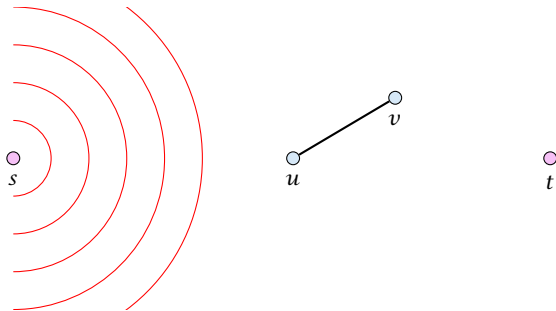
What is the probability that an edge (u, v) is in the cut?



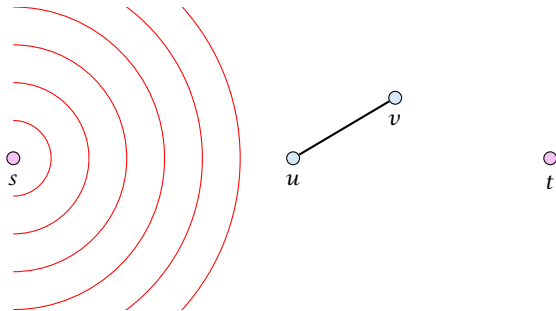
What is the probability that an edge (u, v) is in the cut?



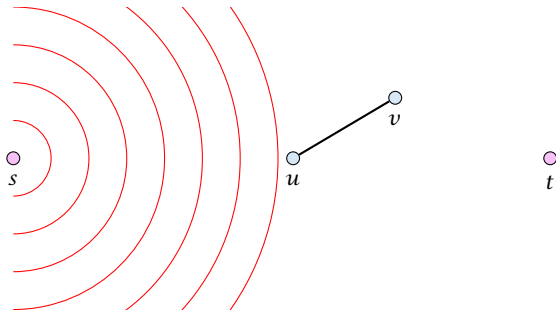
What is the probability that an edge (u, v) is in the cut?



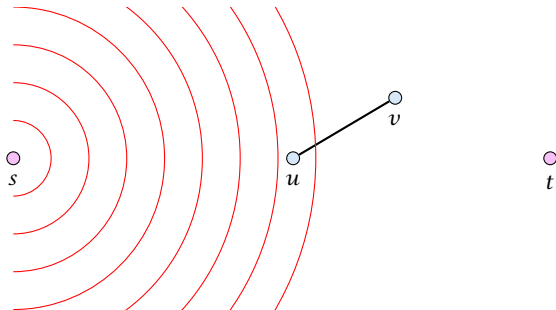
What is the probability that an edge (u, v) is in the cut?



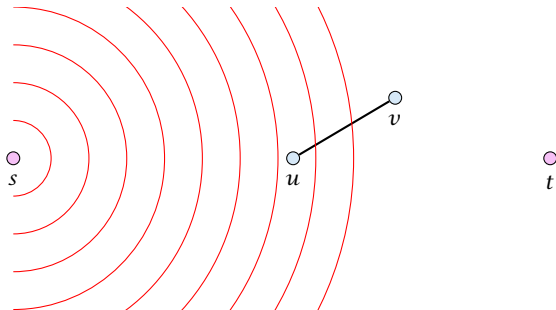
What is the probability that an edge (u, v) is in the cut?



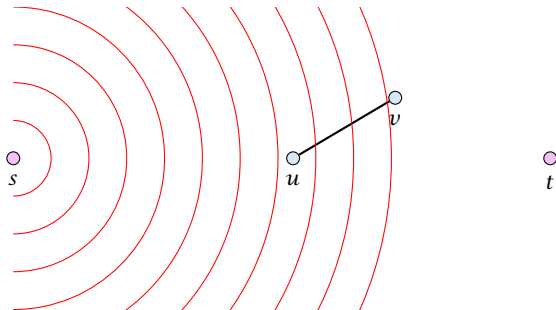
What is the probability that an edge (u, v) is in the cut?



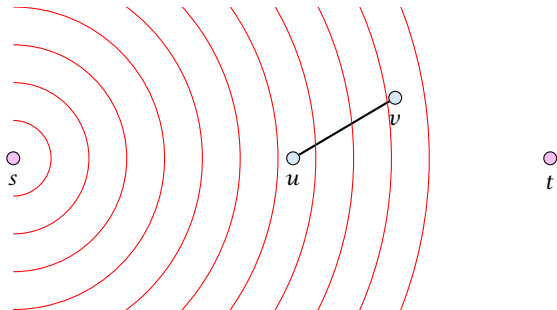
What is the probability that an edge (u, v) is in the cut?



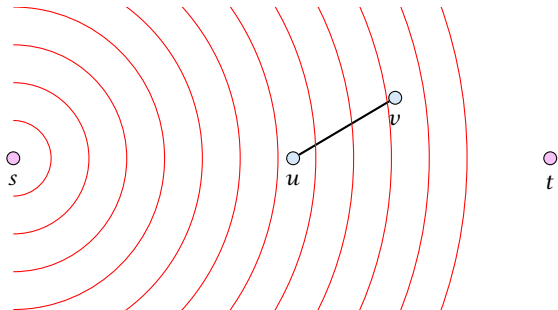
What is the probability that an edge (u, v) is in the cut?



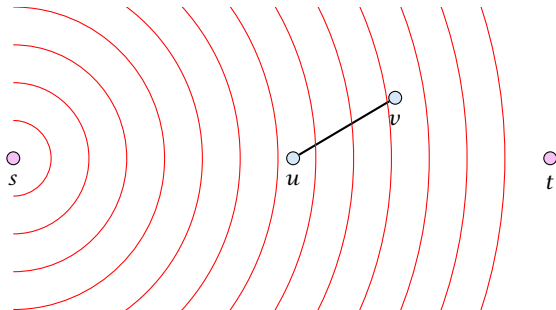
What is the probability that an edge (u, v) is in the cut?



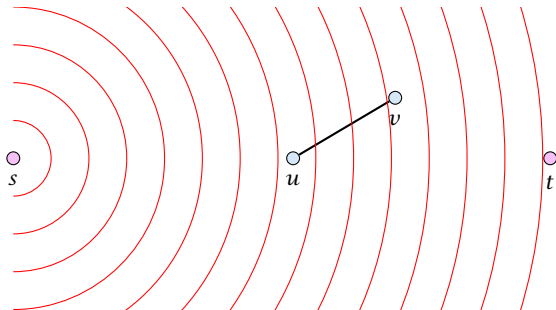
What is the probability that an edge (u, v) is in the cut?



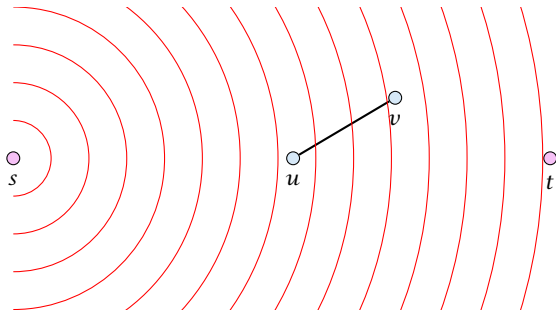
What is the probability that an edge (u, v) is in the cut?



What is the probability that an edge (u, v) is in the cut?



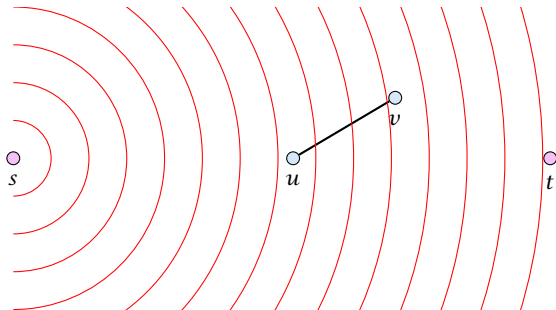
What is the probability that an edge (u, v) is in the cut?



► assume wlog. $d(s, u) \leq d(s, v)$

$\Pr[e \text{ is cut}]$

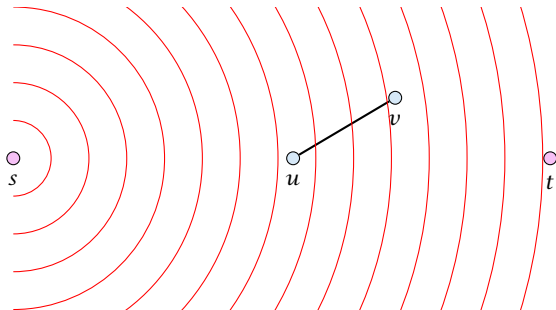
What is the probability that an edge (u, v) is in the cut?



► assume wlog. $d(s, u) \leq d(s, v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s, u), d(s, v)]]$$

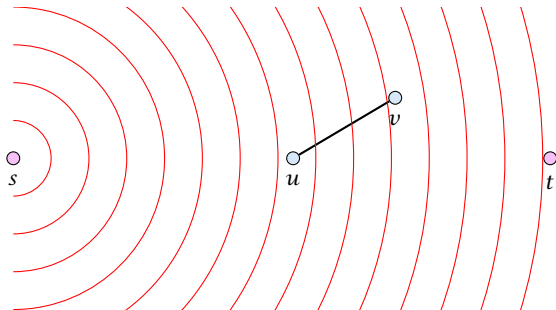
What is the probability that an edge (u, v) is in the cut?



► assume wlog. $d(s, u) \leq d(s, v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s, u), d(s, v)]] \leq \frac{d(s, v) - d(s, u)}{1 - 0}$$

What is the probability that an edge (u, v) is in the cut?



► assume wlog. $d(s, u) \leq d(s, v)$

$$\begin{aligned}\Pr[e \text{ is cut}] &= \Pr[r \in [d(s, u), d(s, v)]] \leq \frac{d(s, v) - d(s, u)}{1 - 0} \\ &\leq \ell_e\end{aligned}$$

What is the expected size of a cut?

$$\begin{aligned} \mathbb{E}[\text{size of cut}] &= \mathbb{E}\left[\sum_e c(e) \Pr[e \text{ is cut}]\right] \\ &\leq \sum_e c(e) \ell_e \end{aligned}$$

On the other hand:

$$\sum_e c(e) \ell_e \leq \text{size of mincut}$$

as the ℓ_e are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

What is the expected size of a cut?

$$\begin{aligned} \mathbb{E}[\text{size of cut}] &= \mathbb{E}\left[\sum_e c(e) \Pr[e \text{ is cut}]\right] \\ &\leq \sum_e c(e) \ell_e \end{aligned}$$

On the other hand:

$$\sum_e c(e) \ell_e \leq \text{size of mincut}$$

as the ℓ_e are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

What is the expected size of a cut?

$$\begin{aligned} \mathbb{E}[\text{size of cut}] &= \mathbb{E}\left[\sum_e c(e) \Pr[e \text{ is cut}]\right] \\ &\leq \sum_e c(e) \ell_e \end{aligned}$$

On the other hand:

$$\sum_e c(e) \ell_e \leq \text{size of mincut}$$

as the ℓ_e are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

Minimum Multicut:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i , $i = 1, \dots, k$, and a capacity function $c : E \rightarrow \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all s_i - t_i pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{ll} \min & \sum_e c(e) \ell_e \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i \quad \sum_{e \in P} \ell_e \geq 1 \\ & \forall e \in E \quad \ell_e \in \{0, 1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path P between s_i and t_i .

Minimum Multicut:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i , $i = 1, \dots, k$, and a capacity function $c : E \rightarrow \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all s_i - t_i pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{ll} \min & \sum_e c(e) \ell_e \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i \quad \sum_{e \in P} \ell_e \geq 1 \\ & \forall e \in E \quad \ell_e \in \{0, 1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path P between s_i and t_i .

Minimum Multicut:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i , $i = 1, \dots, k$, and a capacity function $c : E \rightarrow \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all s_i - t_i pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{ll} \min & \sum_e c(e) \ell_e \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i \quad \sum_{e \in P} \ell_e \geq 1 \\ & \forall e \in E \quad \ell_e \in \{0, 1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path P between s_i and t_i .

Re-using the analysis for the single-commodity case is difficult.

$$\Pr[e \text{ is cut}] \leq ?$$

- ▶ If for some R the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- ▶ However, this cannot be guaranteed.

Re-using the analysis for the single-commodity case is difficult.

$$\Pr[e \text{ is cut}] \leq ?$$

- ▶ If for some R the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- ▶ However, this cannot be guaranteed.

Re-using the analysis for the single-commodity case is difficult.

$$\Pr[e \text{ is cut}] \leq ?$$

- ▶ If for some R the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- ▶ However, this cannot be guaranteed.

- ▶ Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- ▶ Replace the graph G by a graph G' , where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ .
- ▶ Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \leq z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
1:  $z \leftarrow 0$ 
2: repeat
3:     flip a coin ( $\Pr[\text{heads}] = p$ )
4:      $z \leftarrow z + 1$ 
5: until heads
6: return  $B(s_i, z)$ 
```

- ▶ Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- ▶ Replace the graph G by a graph G' , where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ .
- ▶ Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \leq z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
1:  $z \leftarrow 0$ 
2: repeat
3:   flip a coin ( $\Pr[\text{heads}] = p$ )
4:    $z \leftarrow z + 1$ 
5: until heads
6: return  $B(s_i, z)$ 
```

- ▶ Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- ▶ Replace the graph G by a graph G' , where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ .
- ▶ Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \leq z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
1:  $z \leftarrow 0$ 
2: repeat
3:     flip a coin ( $\Pr[\text{heads}] = p$ )
4:      $z \leftarrow z + 1$ 
5: until heads
6: return  $B(s_i, z)$ 
```

- ▶ Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- ▶ Replace the graph G by a graph G' , where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ .
- ▶ Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \leq z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
1:  $z \leftarrow 0$ 
2: repeat
3:   flip a coin ( $\Pr[\text{heads}] = p$ )
4:    $z \leftarrow z + 1$ 
5: until heads
6: return  $B(s_i, z)$ 
```

Algorithm 1 Multicut(G')

```
1: while  $\exists s_i-t_i$  pair in  $G'$  do  
2:    $C \leftarrow \text{RegionGrowing}(s_i, p)$   
3:    $G' = G' \setminus C$  // cuts edges leaving  $C$   
4: return  $B(s_i, z)$ 
```

- ▶ probability of cutting an edge is only p
- ▶ a source either does not reach an edge during Region Growing; then it is not cut
- ▶ if it reaches the edge then it either **cuts** the edge or **protects** the edge from being cut by other sources
- ▶ if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

```
1: while  $\exists s_i-t_i$  pair in  $G'$  do  
2:    $C \leftarrow \text{RegionGrowing}(s_i, p)$   
3:    $G' = G' \setminus C$  // cuts edges leaving  $C$   
4: return  $B(s_i, z)$ 
```

- ▶ probability of cutting an edge is only p
- ▶ a source either does not reach an edge during Region Growing; then it is not cut
- ▶ if it reaches the edge then it either **cuts** the edge or **protects** the edge from being cut by other sources
- ▶ if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

```
1: while  $\exists s_i-t_i$  pair in  $G'$  do  
2:    $C \leftarrow \text{RegionGrowing}(s_i, p)$   
3:    $G' = G' \setminus C$  // cuts edges leaving  $C$   
4: return  $B(s_i, z)$ 
```

- ▶ probability of cutting an edge is only p
- ▶ a source either does not reach an edge during Region Growing; then it is not cut
- ▶ if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- ▶ if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

```
1: while  $\exists s_i-t_i$  pair in  $G'$  do  
2:    $C \leftarrow \text{RegionGrowing}(s_i, p)$   
3:    $G' = G' \setminus C$  // cuts edges leaving  $C$   
4: return  $B(s_i, z)$ 
```

- ▶ probability of cutting an edge is only p
- ▶ a source either does not reach an edge during Region Growing; then it is not cut
- ▶ if it reaches the edge then it either **cuts** the edge or **protects** the edge from being cut by other sources
- ▶ if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

```
1: while  $\exists s_i-t_i$  pair in  $G'$  do  
2:    $C \leftarrow \text{RegionGrowing}(s_i, p)$   
3:    $G' = G' \setminus C$  // cuts edges leaving  $C$   
4: return  $B(s_i, z)$ 
```

- ▶ probability of cutting an edge is only p
- ▶ a source either does not reach an edge during Region Growing; then it is not cut
- ▶ if it reaches the edge then it either **cuts** the edge or **protects** the edge from being cut by other sources
- ▶ if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most **OPT**.

Problem:

We may not cut all source-target pairs.

A component that we remove may contain an s_i-t_i pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

Problem:

We may not cut all source-target pairs.

A component that we remove may contain an s_i-t_i pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

Problem:

We may not cut all source-target pairs.

A component that we remove may contain an s_i-t_i pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

- ▶ choose $p = 6 \ln k \cdot \delta$
- ▶ we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- ▶ we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \leq (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \leq e^{-\frac{p}{2\delta}} \leq \frac{1}{k^3}$$

- ▶ Hence,

$$\Pr[\exists i \text{ that is not successful}] \leq \frac{1}{k^2}$$

- ▶ choose $p = 6 \ln k \cdot \delta$
- ▶ we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- ▶ we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \leq (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \leq e^{-\frac{p}{2\delta}} \leq \frac{1}{k^3}$$

- ▶ Hence,

$$\Pr[\exists i \text{ that is not successful}] \leq \frac{1}{k^2}$$

- ▶ choose $p = 6 \ln k \cdot \delta$
- ▶ we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- ▶ we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \leq (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \leq e^{-\frac{p}{2\delta}} \leq \frac{1}{k^3}$$

▶ Hence,

$$\Pr[\exists i \text{ that is not successful}] \leq \frac{1}{k^2}$$

- ▶ choose $p = 6 \ln k \cdot \delta$
- ▶ we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- ▶ we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \leq (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \leq e^{-\frac{p}{2\delta}} \leq \frac{1}{k^3}$$

- ▶ Hence,

$$\Pr[\exists i \text{ that is not successful}] \leq \frac{1}{k^2}$$

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} E[\text{cutsize} \mid \text{succ.}] &= \frac{E[\text{cutsize}] - \Pr[\text{no succ.}] \cdot E[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\leq \frac{E[\text{cutsize}]}{\Pr[\text{success}]} \leq \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \leq 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} E[\text{cutsize} \mid \text{succ.}] &= \frac{E[\text{cutsize}] - \Pr[\text{no succ.}] \cdot E[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\leq \frac{E[\text{cutsize}]}{\Pr[\text{success}]} \leq \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \leq 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} E[\text{cutsize} \mid \text{succ.}] &= \frac{E[\text{cutsize}] - \Pr[\text{no succ.}] \cdot E[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\leq \frac{E[\text{cutsize}]}{\Pr[\text{success}]} \leq \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \leq 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} E[\text{cutsize} \mid \text{succ.}] &= \frac{E[\text{cutsize}] - \Pr[\text{no succ.}] \cdot E[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\leq \frac{E[\text{cutsize}]}{\Pr[\text{success}]} \leq \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \leq 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} E[\text{cutsize} \mid \text{succ.}] &= \frac{E[\text{cutsize}] - \Pr[\text{no succ.}] \cdot E[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\leq \frac{E[\text{cutsize}]}{\Pr[\text{success}]} \leq \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \leq 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned} E[\text{cutsize}] &= \Pr[\text{success}] \cdot E[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot E[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} E[\text{cutsize} \mid \text{succ.}] &= \frac{E[\text{cutsize}] - \Pr[\text{no succ.}] \cdot E[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\leq \frac{E[\text{cutsize}]}{\Pr[\text{success}]} \leq \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \leq 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: **success** means all source-target pairs separated

We assume $k \geq 2$.

If we are not successful we simply perform a trivial k -approximation.

This only increases the expected cost by at most $\frac{1}{k^2} \cdot k\text{OPT} \leq \text{OPT}/k$.

Hence, our final cost is $\mathcal{O}(\ln k) \cdot \text{OPT}$ in expectation.

Traveling Salesman

Given a set of cities $\{1, \dots, n\}$ and a symmetric matrix $C = (c_{ij})$, $c_{ij} \geq 0$ that specifies for every pair $(i, j) \in [n] \times [n]$ the cost for travelling from city i to city j . Find a permutation π of the cities such that the round-trip cost

$$c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$$

is minimized.

Traveling Salesman

Theorem 94

There does not exist an $O(2^n)$ -approximation algorithm for TSP.

Hamiltonian Cycle:

For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G .

Given an instance to HAMILTONIAN, we create an instance for TSP.

If $|V| = 1$, then set c_{ij} to 0. Otherwise, set c_{ij} to 1. This way, the instance has polynomial size.

There exists a Hamiltonian Cycle if and only if there exists a tour with cost $|V|$. Our tiny tour has cost strictly larger than $|V|$.

No $O(2^n)$ -approximation algorithm could decide that there is no proper tour, cannot exist unless P=NP.

Traveling Salesman

Theorem 94

There does not exist an $O(2^n)$ -approximation algorithm for TSP.

Hamiltonian Cycle:

For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G .

Traveling Salesman

Theorem 94

There does not exist an $O(2^n)$ -approximation algorithm for TSP.

Hamiltonian Cycle:

For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G .

- ▶ Given an instance to HAMPATH we create an instance for TSP.
- ▶ If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- ▶ There exists a Hamiltonian Path iff there exists a tour with cost n . Otw. any tour has cost strictly larger than $n2^n$.
- ▶ An $O(2^n)$ -approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 94

There does not exist an $O(2^n)$ -approximation algorithm for TSP.

Hamiltonian Cycle:

For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G .

- ▶ Given an instance to HAMPATH we create an instance for TSP.
- ▶ If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- ▶ There exists a Hamiltonian Path iff there exists a tour with cost n . Otw. any tour has cost strictly larger than $n2^n$.
- ▶ An $O(2^n)$ -approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 94

There does not exist an $O(2^n)$ -approximation algorithm for TSP.

Hamiltonian Cycle:

For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G .

- ▶ Given an instance to HAMPATH we create an instance for TSP.
- ▶ If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- ▶ There exists a Hamiltonian Path iff there exists a tour with cost n . Otw. any tour has cost strictly larger than $n2^n$.
- ▶ An $O(2^n)$ -approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 94

There does not exist an $O(2^n)$ -approximation algorithm for TSP.

Hamiltonian Cycle:

For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G .

- ▶ Given an instance to HAMPATH we create an instance for TSP.
- ▶ If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- ▶ There exists a Hamiltonian Path iff there exists a tour with cost n . Otw. any tour has cost strictly larger than $n2^n$.
- ▶ An $O(2^n)$ -approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Metric Traveling Salesman

In the metric version we assume for every triple $i, j, k \in \{1, \dots, n\}$

$$c_{ik} \leq c_{ij} + c_{jk} .$$

It is convenient to view the input as a complete undirected graph $G = (V, E)$, where c_{ij} for an edge (i, j) defines the distance between nodes i and j .

Metric Traveling Salesman

In the metric version we assume for every triple $i, j, k \in \{1, \dots, n\}$

$$c_{ik} \leq c_{ij} + c_{jk} .$$

It is convenient to view the input as a complete undirected graph $G = (V, E)$, where c_{ij} for an edge (i, j) defines the distance between nodes i and j .

Lemma 95

The cost $\text{OPT}_{\text{TSP}}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\text{OPT}_{\text{MST}}(G)$ of a minimum spanning tree in G .

Proof:

- 1. Let T be the minimum spanning tree of G .
- 2. Let OPT_{TSP} be an optimum traveling salesman tour.
- 3. Let OPT_{MST} be the weight of a minimum spanning tree.
- 4. Let OPT_{TSP} be the cost of an optimum traveling salesman tour.

Lemma 95

The cost $\text{OPT}_{\text{TSP}}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\text{OPT}_{\text{MST}}(G)$ of a minimum spanning tree in G .

Proof:

- ▶ Take the optimum TSP-tour.
- ▶ Delete one edge.
- ▶ This gives a spanning tree of cost at most $\text{OPT}_{\text{TSP}}(G)$.

TSP: Lower Bound I

Lemma 95

The cost $\text{OPT}_{\text{TSP}}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\text{OPT}_{\text{MST}}(G)$ of a minimum spanning tree in G .

Proof:

- ▶ Take the optimum TSP-tour.
- ▶ Delete one edge.
- ▶ This gives a spanning tree of cost at most $\text{OPT}_{\text{TSP}}(G)$.

Lemma 95

The cost $\text{OPT}_{\text{TSP}}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\text{OPT}_{\text{MST}}(G)$ of a minimum spanning tree in G .

Proof:

- ▶ Take the optimum TSP-tour.
- ▶ Delete one edge.
- ▶ This gives a spanning tree of cost at most $\text{OPT}_{\text{TSP}}(G)$.

TSP: Greedy Algorithm

- ▶ Start with a tour on a subset S containing a single node.
- ▶ Take the node v closest to S . Add it S and expand the existing tour on S to include v .
- ▶ Repeat until all nodes have been processed.

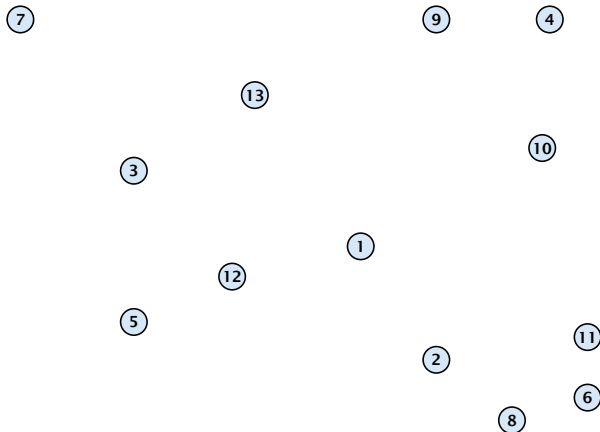
TSP: Greedy Algorithm

- ▶ Start with a tour on a subset S containing a single node.
- ▶ Take the node v closest to S . Add it S and expand the existing tour on S to include v .
- ▶ Repeat until all nodes have been processed.

TSP: Greedy Algorithm

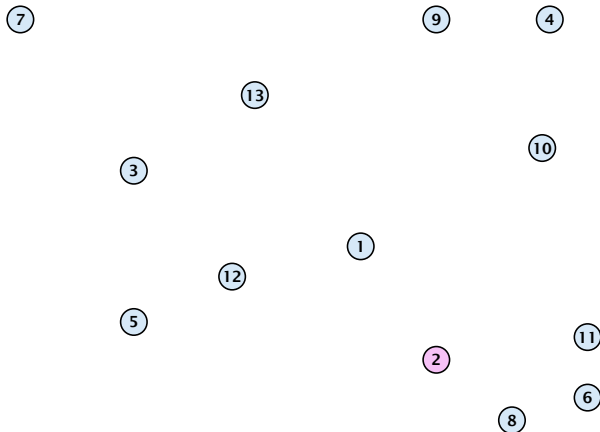
- ▶ Start with a tour on a subset S containing a single node.
- ▶ Take the node v closest to S . Add it S and expand the existing tour on S to include v .
- ▶ Repeat until all nodes have been processed.

TSP: Greedy Algorithm



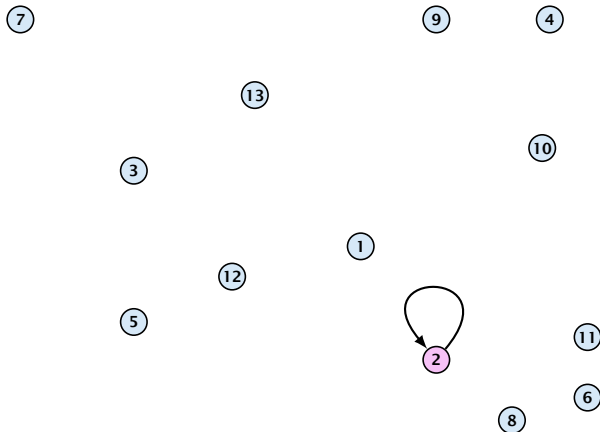
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



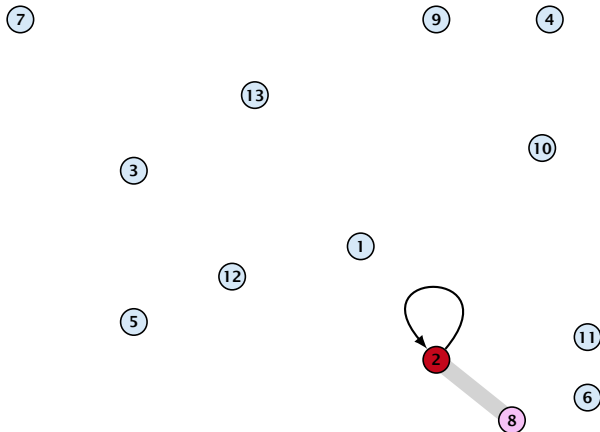
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



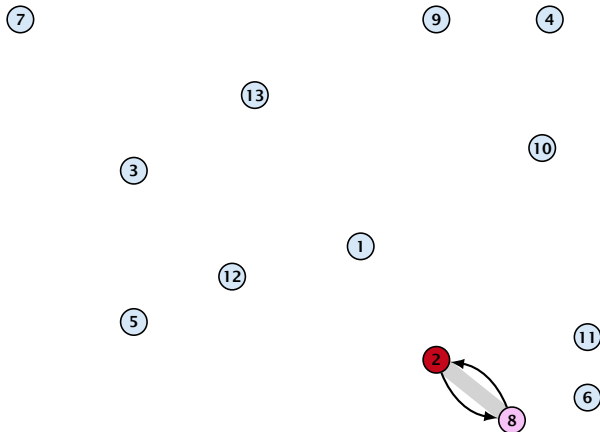
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



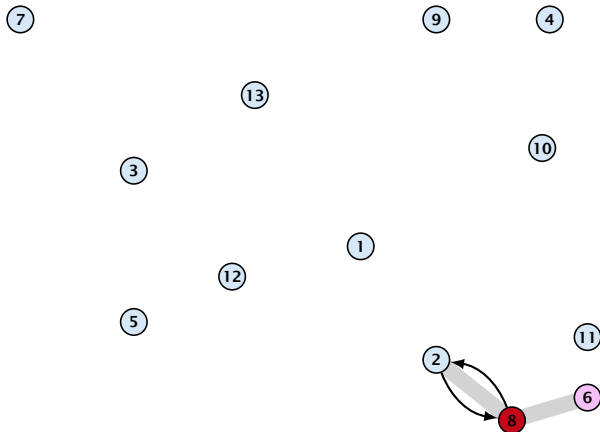
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



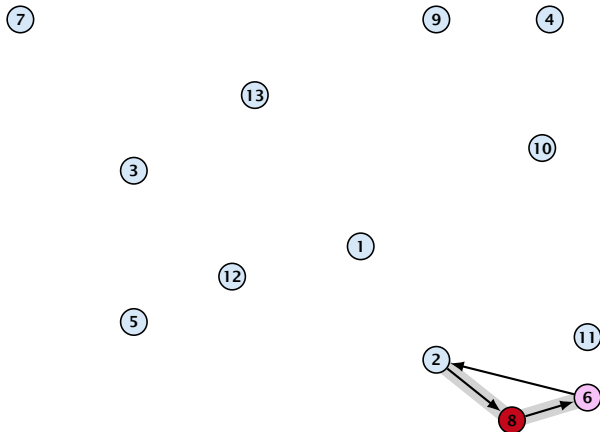
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



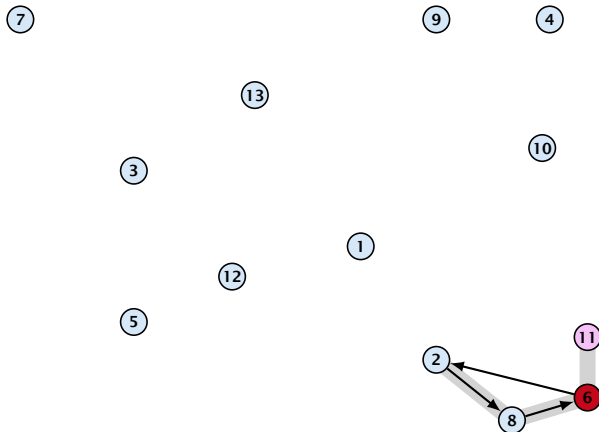
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



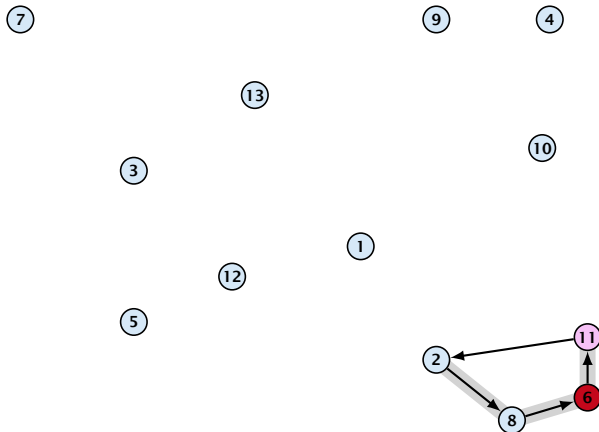
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



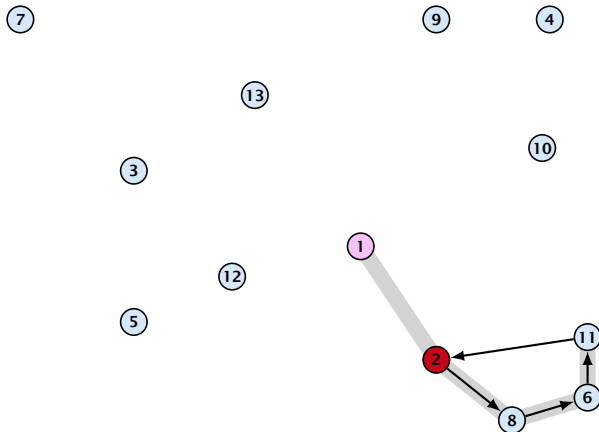
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



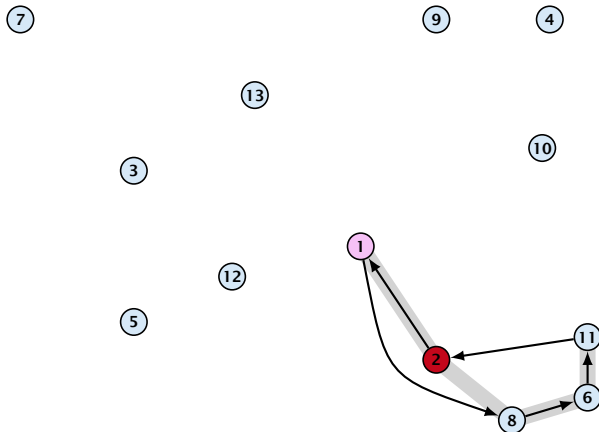
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



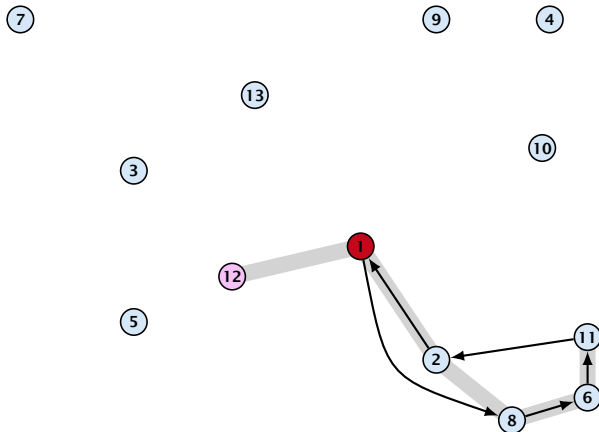
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



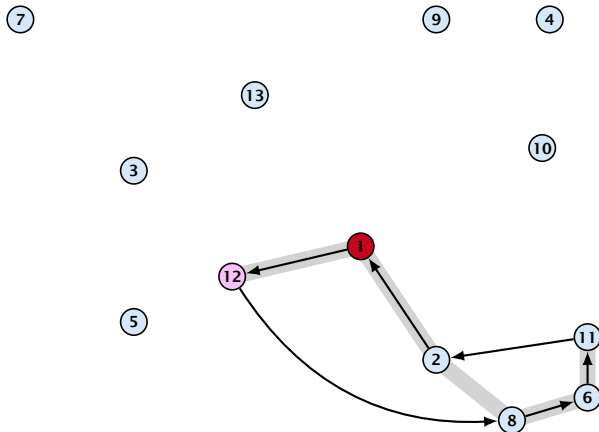
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



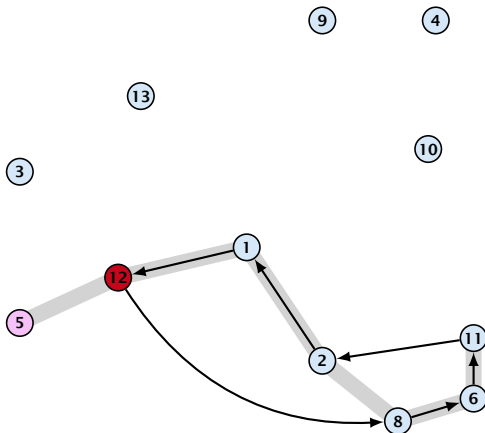
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



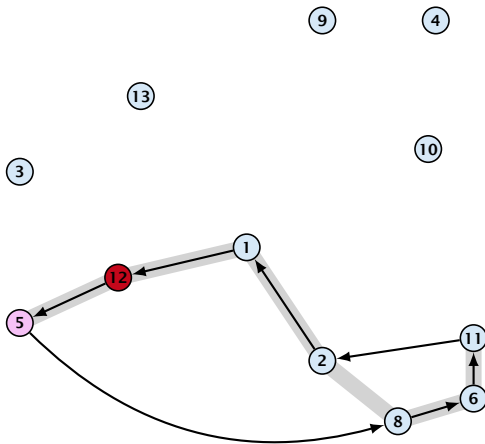
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



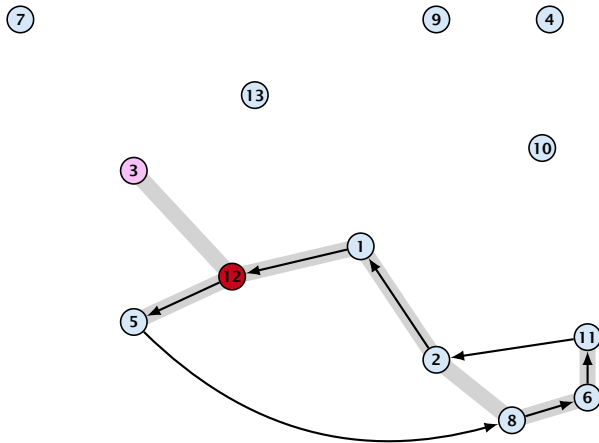
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



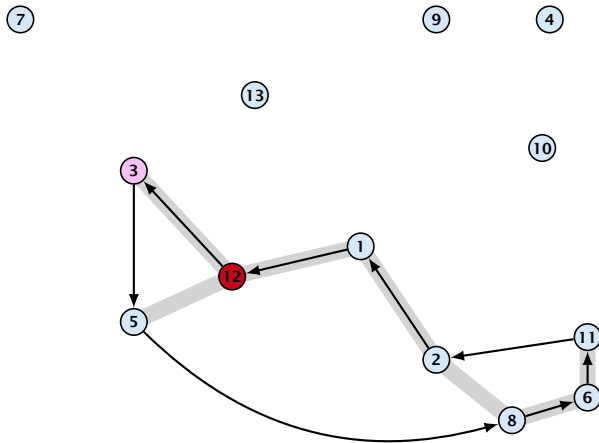
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



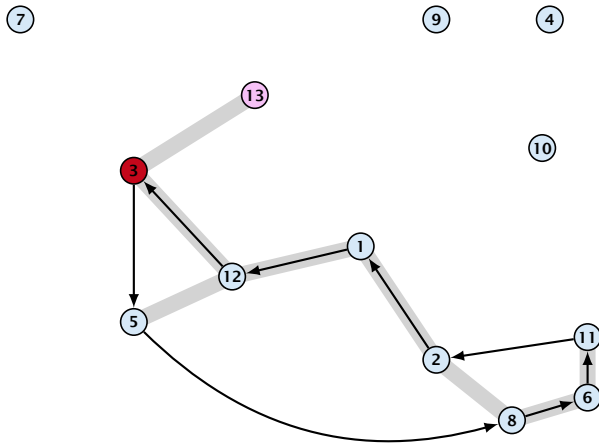
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



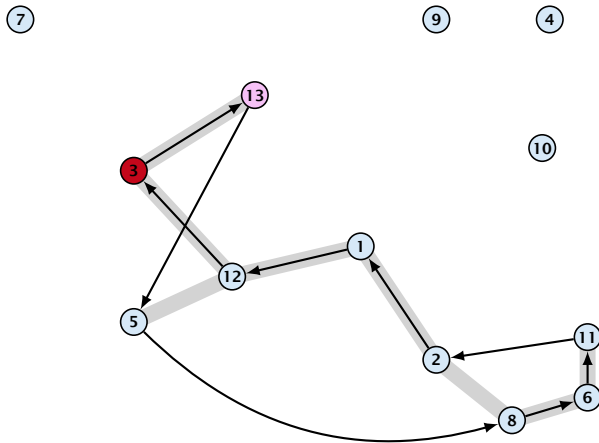
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



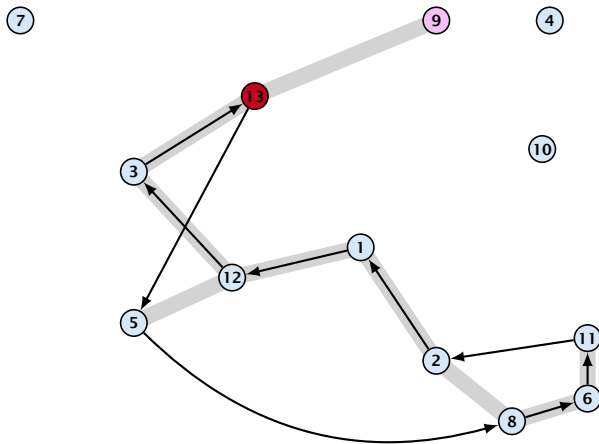
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



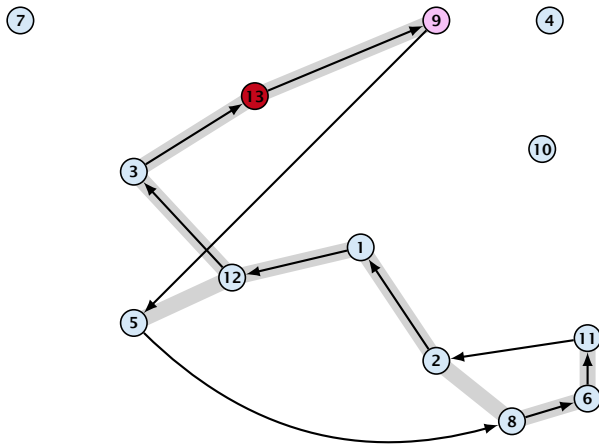
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



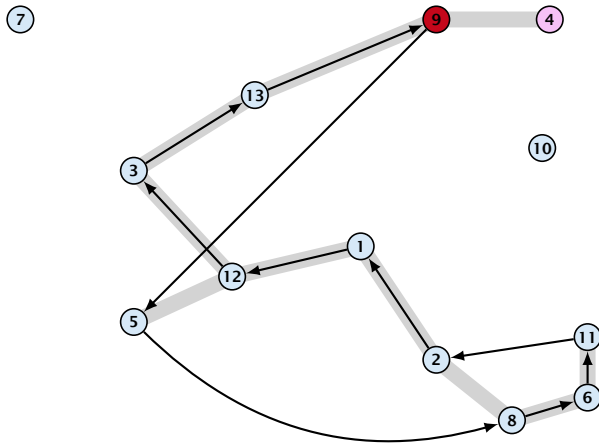
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



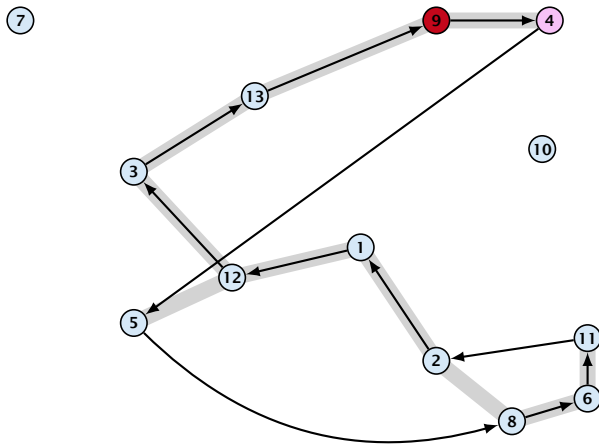
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



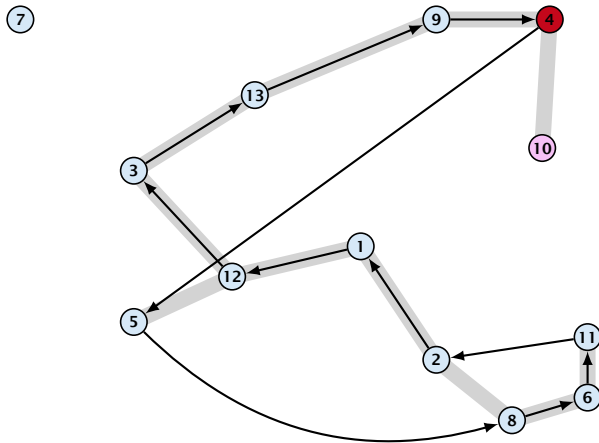
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



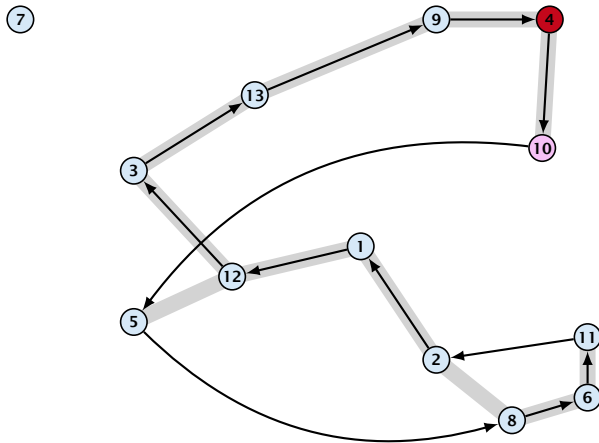
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



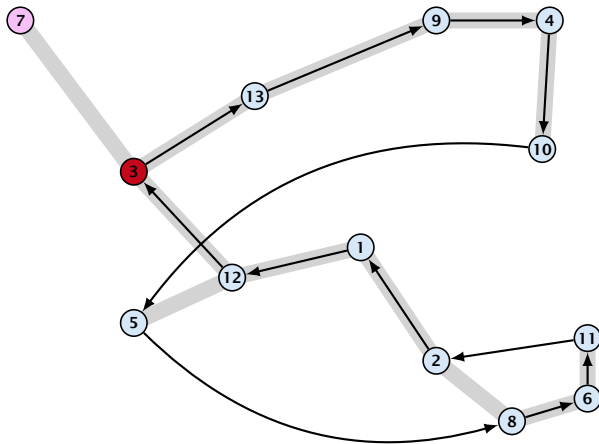
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



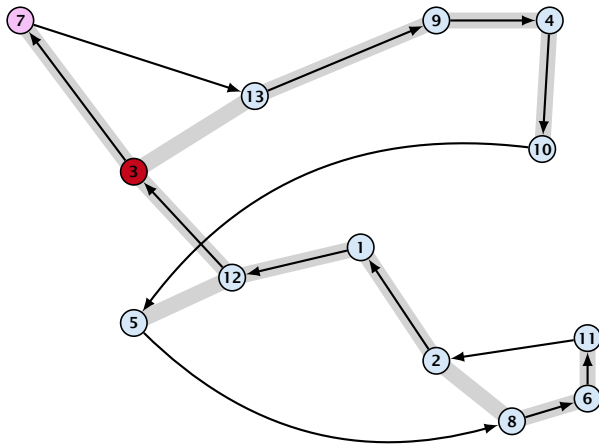
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



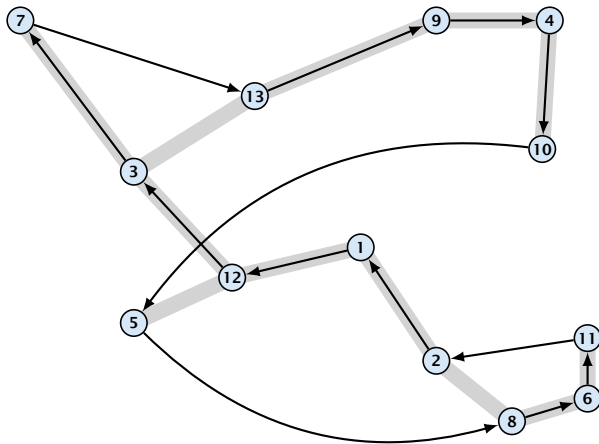
The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

TSP: Greedy Algorithm



The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.



18 TSP

21. Jul. 2023

427/437

TSP: Greedy Algorithm

Lemma 96

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i -th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i .

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i) .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

TSP: Greedy Algorithm

Lemma 96

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i -th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i .

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i) .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

TSP: Greedy Algorithm

Lemma 96

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i -th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i .

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i) .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

TSP: Greedy Algorithm

Lemma 96

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i -th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i .

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i) .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

TSP: Greedy Algorithm

Lemma 96

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i -th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i .

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i) .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

TSP: Greedy Algorithm

Lemma 96

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i -th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i .

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i) .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

TSP: Greedy Algorithm

The edges (s_i, v_i) considered during the Greedy algorithm are exactly the edges considered during PRIMs MST algorithm.

Hence,

$$\sum_i c_{s_i, v_i} = \text{OPT}_{\text{MST}}(G)$$

which with the previous lower bound gives a 2-approximation.

TSP: Greedy Algorithm

The edges (s_i, v_i) considered during the Greedy algorithm are exactly the edges considered during PRIMs MST algorithm.

Hence,

$$\sum_i c_{s_i, v_i} = \text{OPT}_{\text{MST}}(G)$$

which with the previous lower bound gives a 2-approximation.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

Find an Euler tour of G' .

Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.

The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G' .
- Take a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.
- This technique is known as **short cutting** the Euler tour.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

Find an Eulerian graph G' of G .

Then the cost of the TSP-tour is at most the cost of the Euler tour of G' .
The Euler tour of G' only uses the first edge case of a city.

The cost of the TSP-tour is at most the cost of the Euler tour
because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- ▶ Find an Euler tour of G' .
- ▶ Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- ▶ The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- ▶ Find an Euler tour of G' .
- ▶ Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- ▶ The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- ▶ Find an Euler tour of G' .
- ▶ Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- ▶ The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

TSP: A different approach

Suppose that we are given an **Eulerian** graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

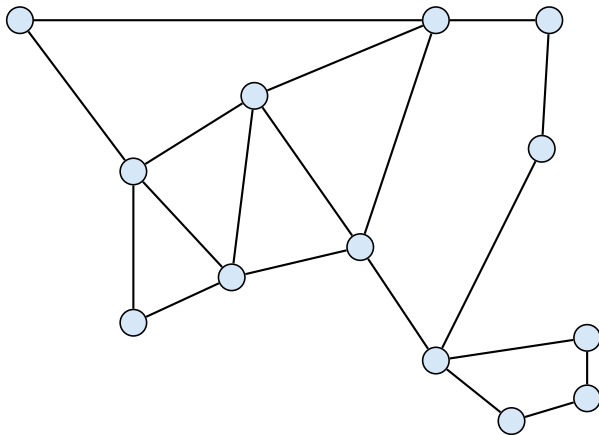
Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

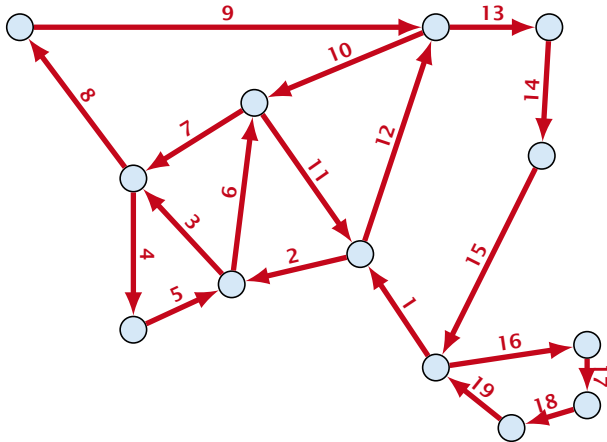
- ▶ Find an Euler tour of G' .
- ▶ Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- ▶ The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

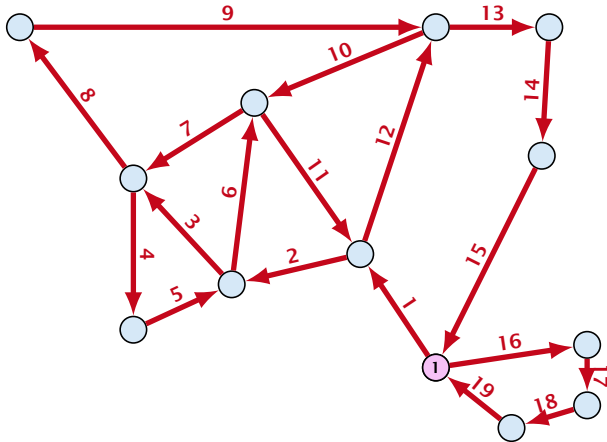
TSP: A different approach



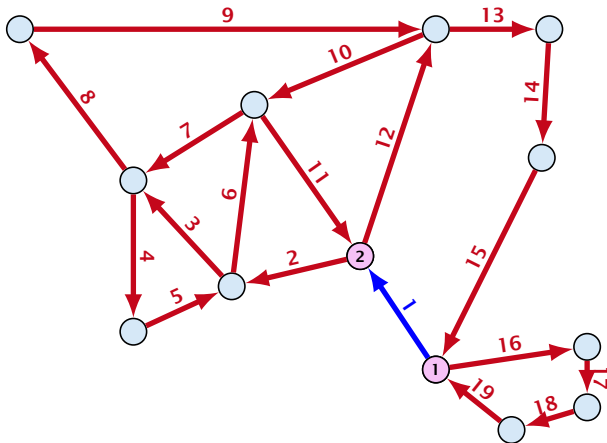
TSP: A different approach



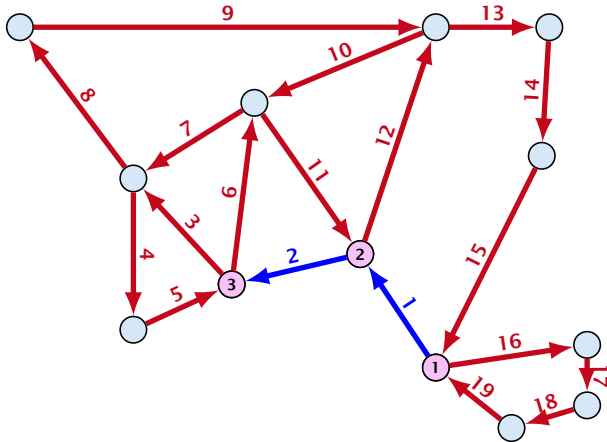
TSP: A different approach



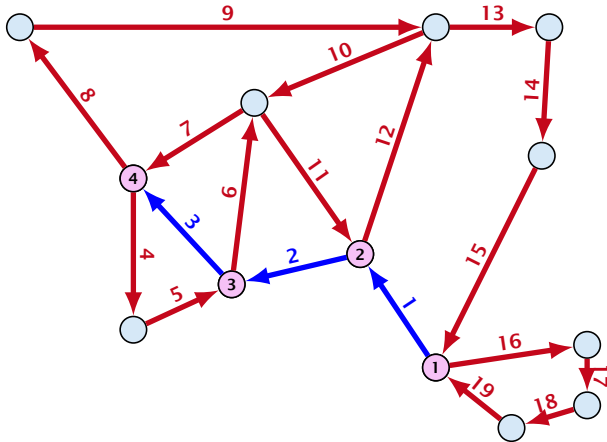
TSP: A different approach



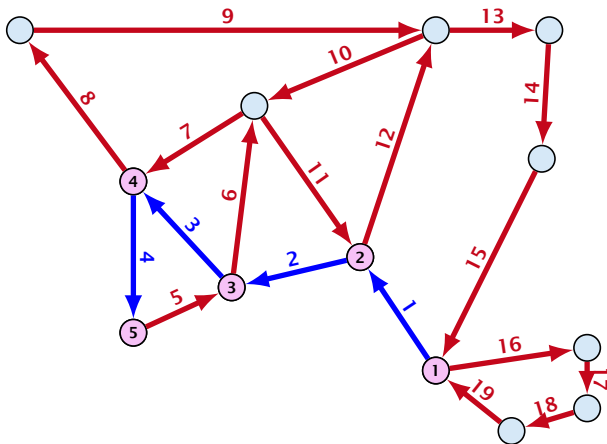
TSP: A different approach



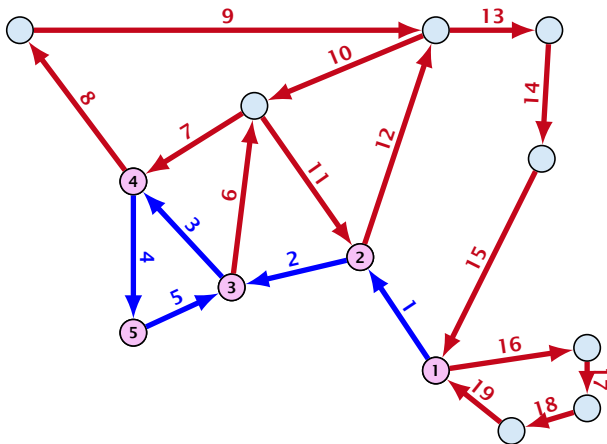
TSP: A different approach



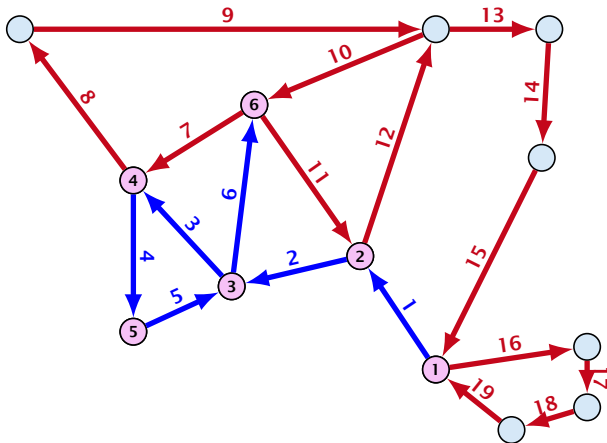
TSP: A different approach



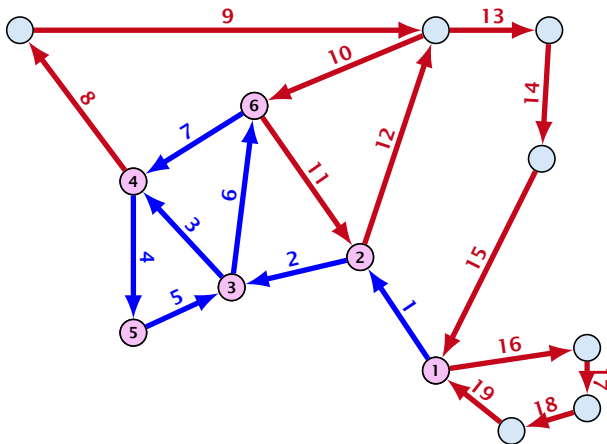
TSP: A different approach



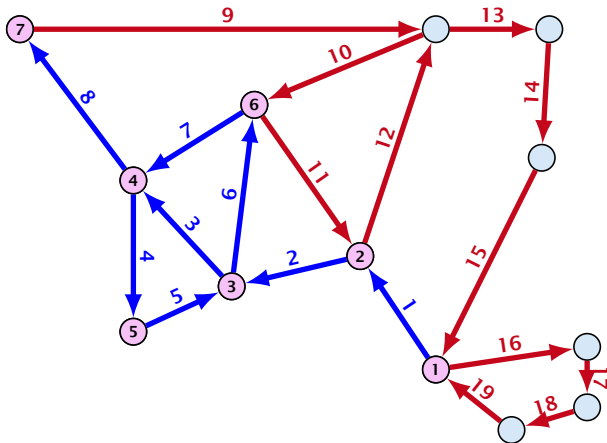
TSP: A different approach



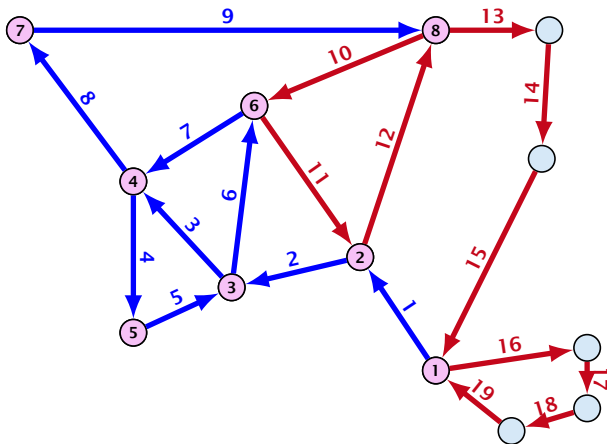
TSP: A different approach



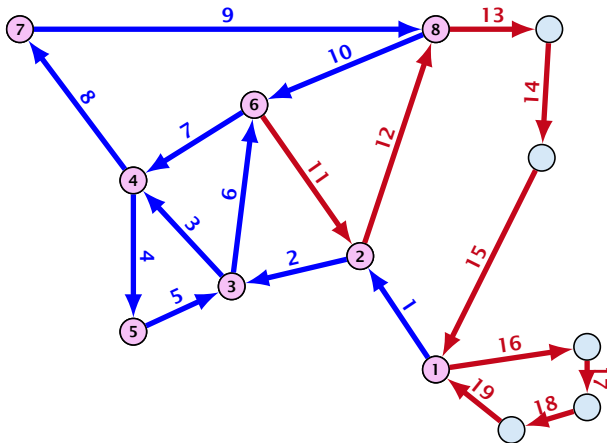
TSP: A different approach



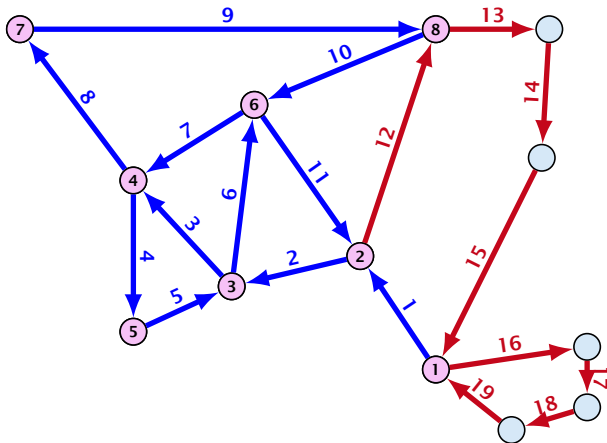
TSP: A different approach



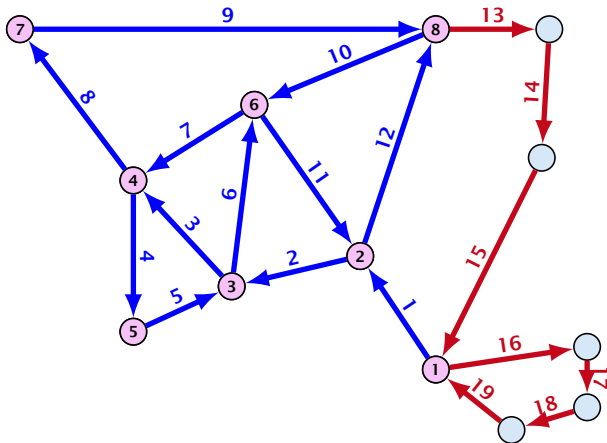
TSP: A different approach



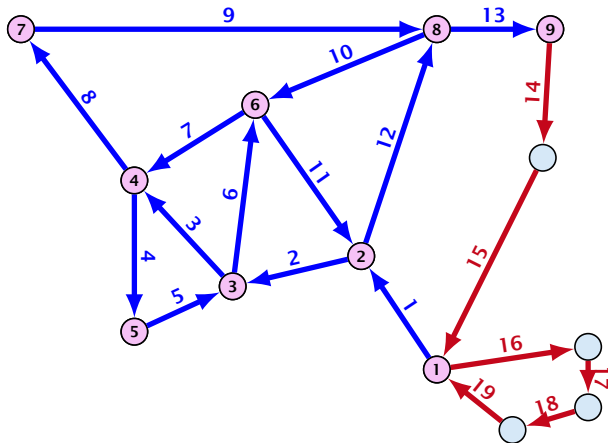
TSP: A different approach



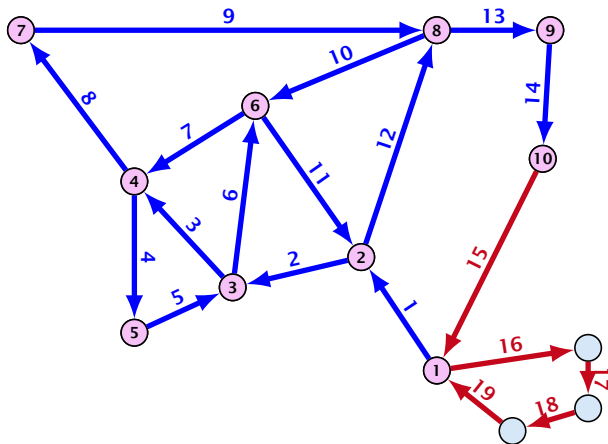
TSP: A different approach



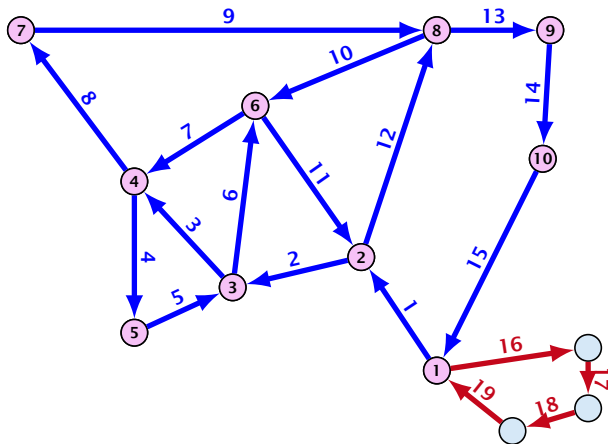
TSP: A different approach



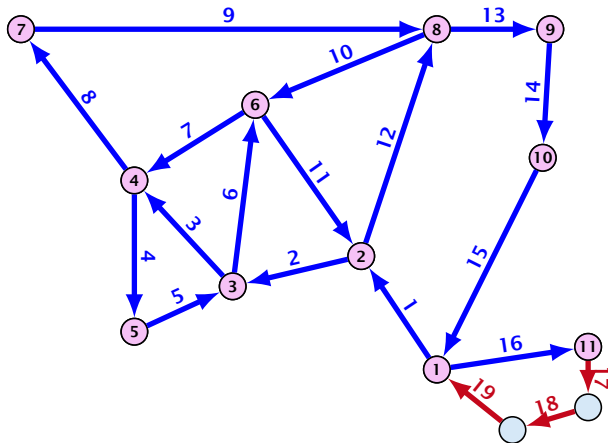
TSP: A different approach



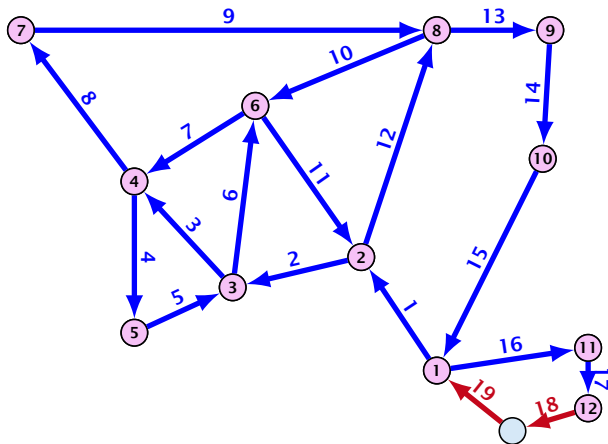
TSP: A different approach



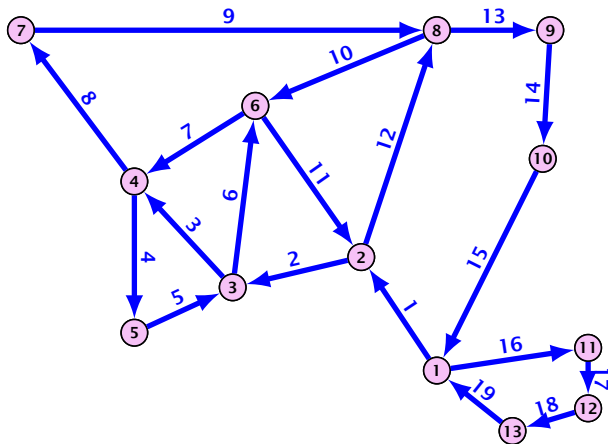
TSP: A different approach



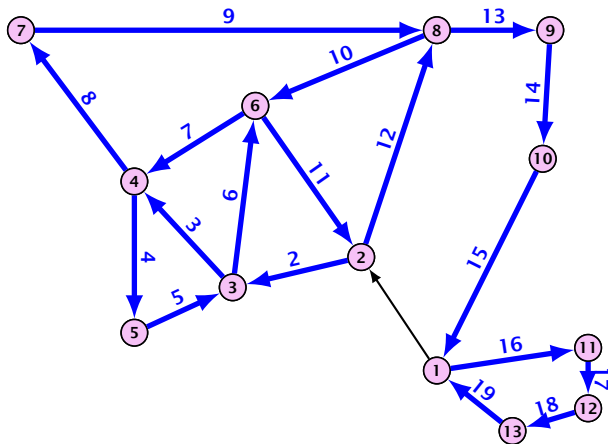
TSP: A different approach



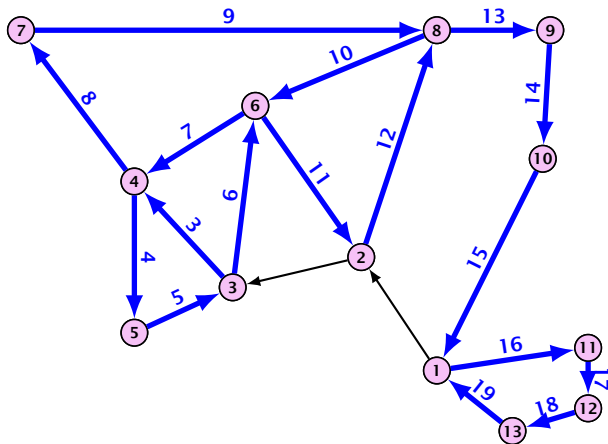
TSP: A different approach



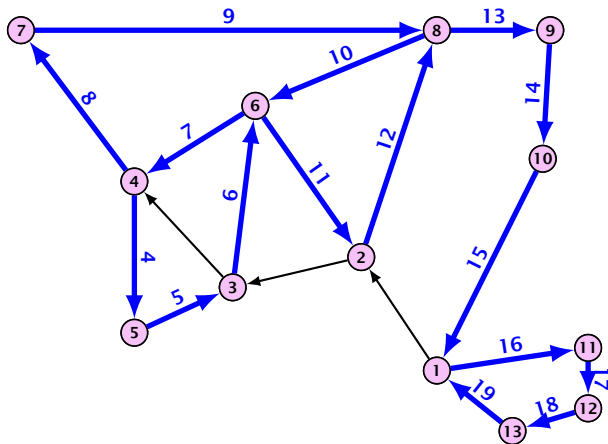
TSP: A different approach



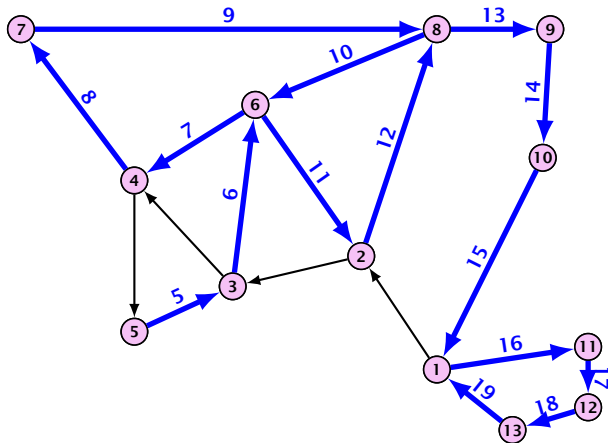
TSP: A different approach



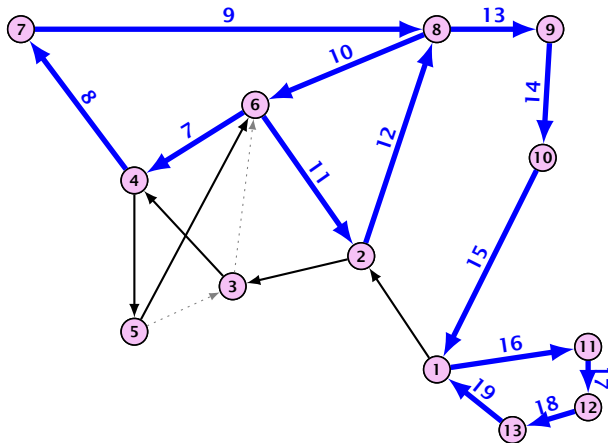
TSP: A different approach



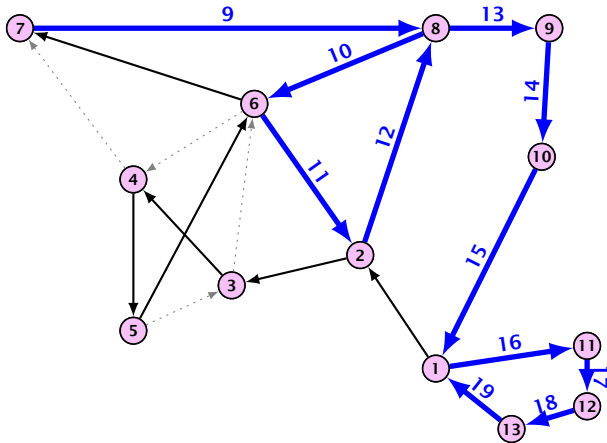
TSP: A different approach



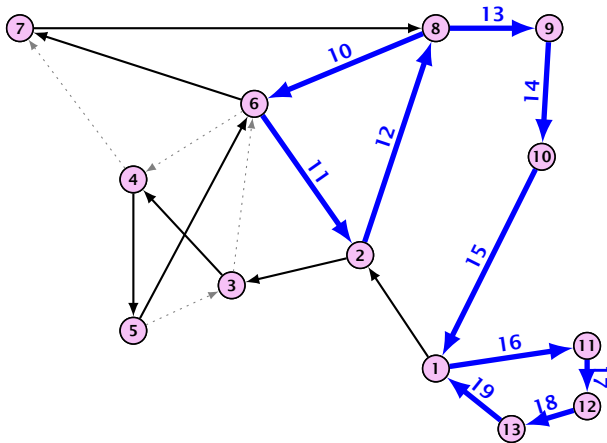
TSP: A different approach



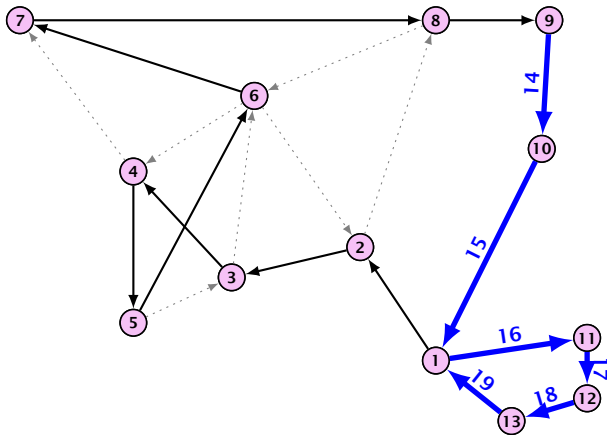
TSP: A different approach



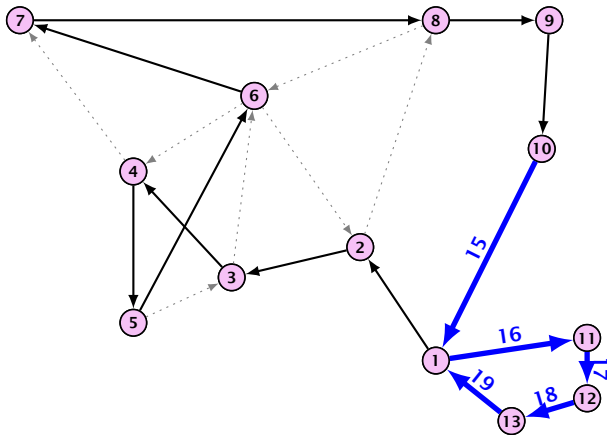
TSP: A different approach



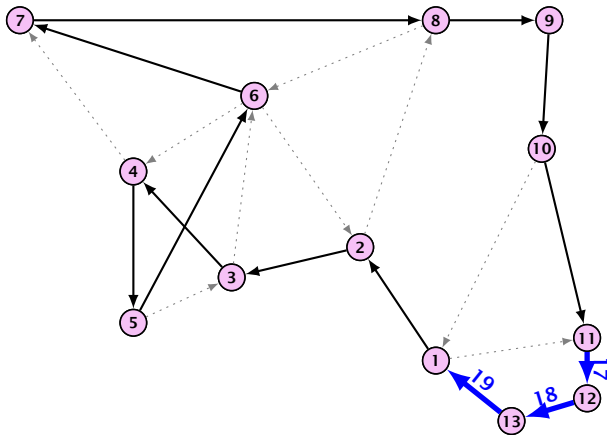
TSP: A different approach



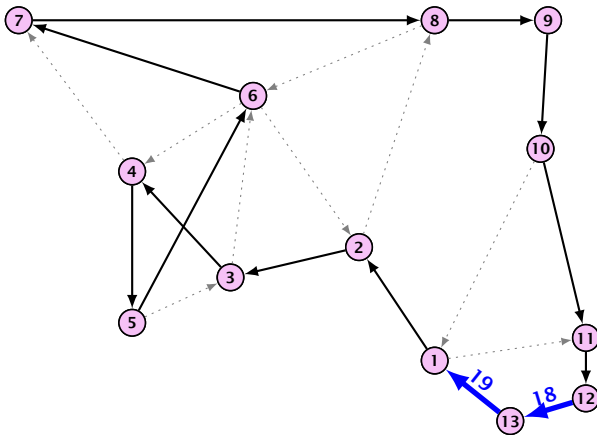
TSP: A different approach



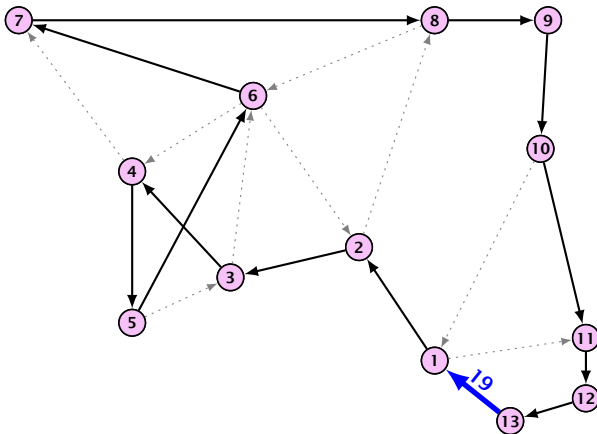
TSP: A different approach



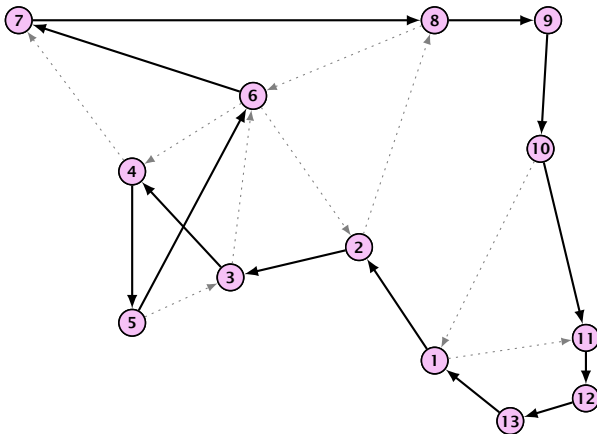
TSP: A different approach



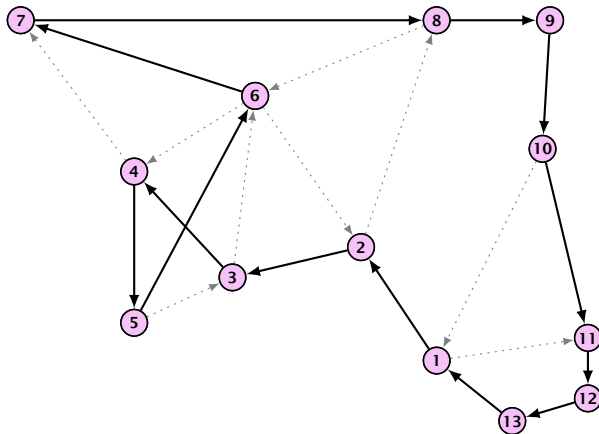
TSP: A different approach



TSP: A different approach



TSP: A different approach



TSP: A different approach

Consider the following graph:

- ▶ Compute an MST of G .
- ▶ Duplicate all edges.

This graph is Eulerian, and the total cost of all edges is at most $2 \cdot \text{OPT}_{\text{MST}}(G)$.

Hence, short-cutting gives a tour of cost no more than $2 \cdot \text{OPT}_{\text{MST}}(G)$ which means we have a 2-approximation.

TSP: A different approach

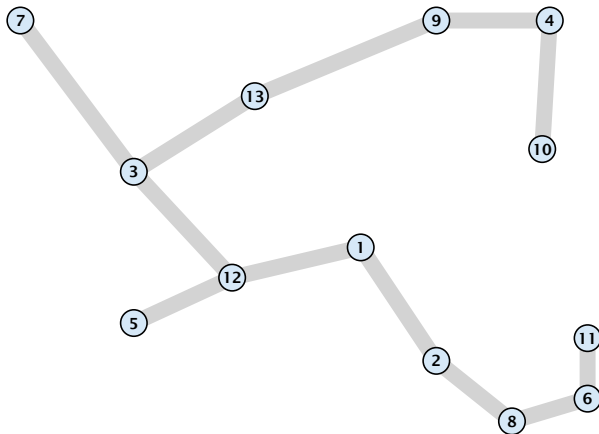
Consider the following graph:

- ▶ Compute an MST of G .
- ▶ Duplicate all edges.

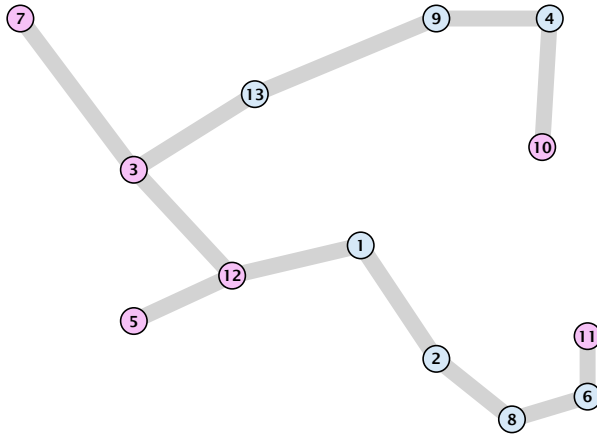
This graph is Eulerian, and the total cost of all edges is at most $2 \cdot \text{OPT}_{\text{MST}}(G)$.

Hence, short-cutting gives a tour of cost no more than $2 \cdot \text{OPT}_{\text{MST}}(G)$ which means we have a 2-approximation.

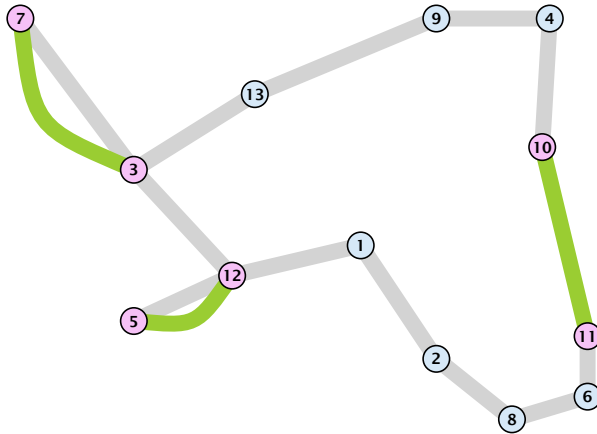
TSP: Can we do better?



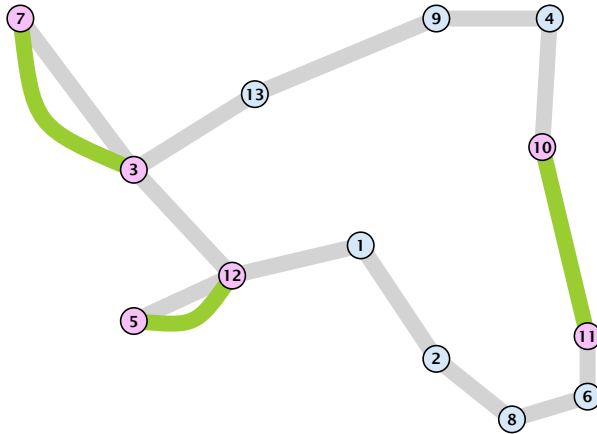
TSP: Can we do better?



TSP: Can we do better?



TSP: Can we do better?



TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\text{OPT}_{\text{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\text{OPT}_{\text{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\text{OPT}_{\text{MST}}(G) + \text{OPT}_{\text{TSP}}(G)/2 \leq \frac{3}{2} \text{OPT}_{\text{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$ -approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\text{OPT}_{\text{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\text{OPT}_{\text{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\text{OPT}_{\text{MST}}(G) + \text{OPT}_{\text{TSP}}(G)/2 \leq \frac{3}{2} \text{OPT}_{\text{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$ -approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\text{OPT}_{\text{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\text{OPT}_{\text{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\text{OPT}_{\text{MST}}(G) + \text{OPT}_{\text{TSP}}(G)/2 \leq \frac{3}{2} \text{OPT}_{\text{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$ -approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\text{OPT}_{\text{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\text{OPT}_{\text{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\text{OPT}_{\text{MST}}(G) + \text{OPT}_{\text{TSP}}(G)/2 \leq \frac{3}{2} \text{OPT}_{\text{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$ -approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\text{OPT}_{\text{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\text{OPT}_{\text{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\text{OPT}_{\text{MST}}(G) + \text{OPT}_{\text{TSP}}(G)/2 \leq \frac{3}{2} \text{OPT}_{\text{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$ -approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\text{OPT}_{\text{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\text{OPT}_{\text{TSP}}(G)/2$.

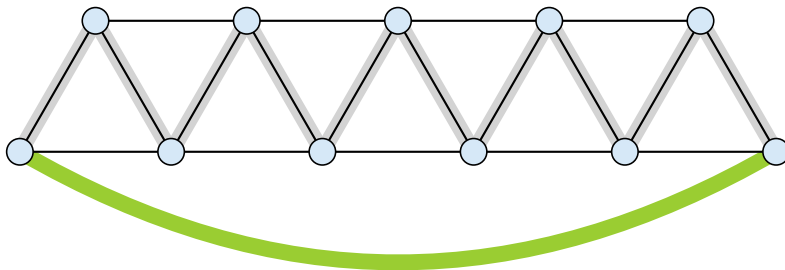
Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\text{OPT}_{\text{MST}}(G) + \text{OPT}_{\text{TSP}}(G)/2 \leq \frac{3}{2} \text{OPT}_{\text{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$ -approximation for metric TSP.

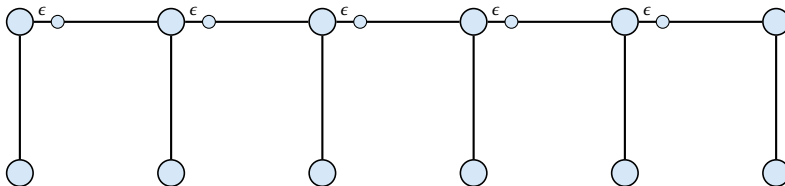
This is the best that is known.

Christofides. Tight Example



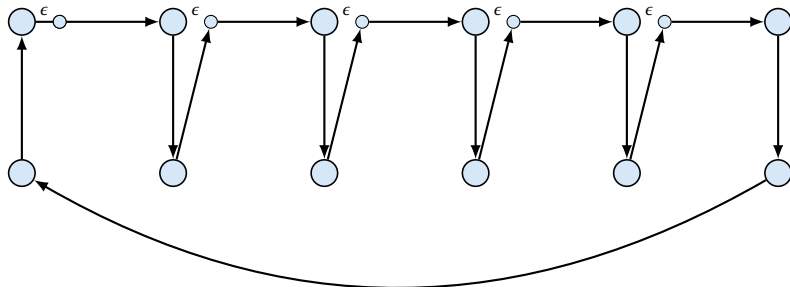
- ▶ optimal tour: n edges.
- ▶ MST: $n - 1$ edges.
- ▶ weight of matching $(n + 1)/2 - 1$
- ▶ MST+matching $\approx 3/2 \cdot n$

Tree shortcutting. Tight Example



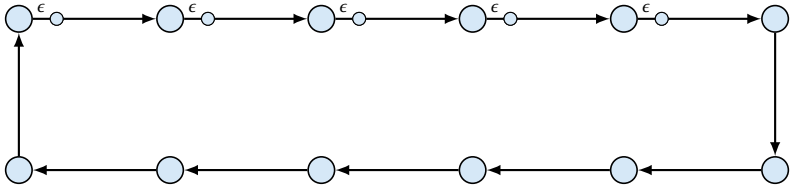
- edges have Euclidean distance.

Tree shortcutting. Tight Example



- ▶ edges have Euclidean distance.

Tree shortcutting. Tight Example



- ▶ edges have Euclidean distance.