Algorithms and Complexity (AC)

Fifth Lecture (9th of October, 2023)

Paul Bouman & Tom van der Zanden (Based on slides by Gerhard Woeginger, Jesper Nederlof and Marie Schmidt)

September – November 2023

LNMB – Landelijk Netwerk Mathematische Besliskunde

Table of contents

Program for this week and the next

Dealing with NP-hard problems: Approximation

Basic definitions ✓

Ad-hoc approaches ✓

LP-based approaches ←

Approximation Schemes

In-approximability

LP-based approaches

LP-based approaches

1. Find an exact ILP formulation
2. Relax integrality constraints (ILP $\rightarrow$ LP)
3. Solve the LP relaxation in polynomial time
4. Round the optimal LP solution to approximate ILP solution (preserving feasibility!)

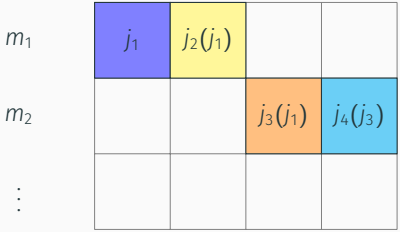
Communication delay scheduling

Communication delay scheduling (COMM-DELAY)

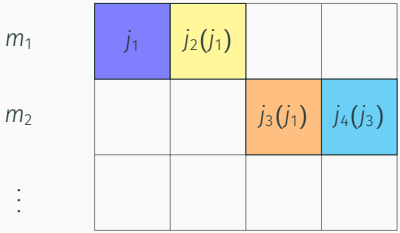
Instance: unit time jobs $J_1, \ldots, J_n$; precedence constraints between some jobs
Goal: find a feasible schedule on n machines that obeys unit communication delays and minimizes makespan

- Unit time jobs: job J_a runs from $S(J_a)$ to $C(J_a) := S(J_a) + 1$
- Precedence constraints = partial order " $\rightarrow$ " on the jobs
- If $J_a \rightarrow J_b$ then $C(J_a) \leq S(J_b) \iff J_a$ must be completed before J_b is started
- Unit communication delay for $J_a \rightarrow J_b$
  - if J_a and J_b run on same machine then $C(J_a) \leq S(J_b)$
  - if J_a and J_b run on different machines then $C(J_a) + 1 \leq S(J_b)$
- Number n of machines is not a bottleneck

Communication delay scheduling



Communication delay scheduling



Notation:

- $\text{Pred}(J_a)$ denotes the set of all predecessors J_b of J_a (with $J_b \to J_a$)
- $\text{Succ}(J_a)$ denotes the set of all successors J_b of J_a (with $J_a \to J_b$)

Observation

At most one predecessor of J_a can complete at $C(J_a) - 1$.
At most one successor of J_a can start at $C(J_a)$.

Modelling idea:

Introduce 0-1-variable x_{ab} that indicates the delay of $J_a \to J_b$

- $x_{ab} = 0$ means that J_b starts directly after J_a on same machine
- $x_{ab} = 1$ means that J_b starts at time $C(J_a) + 1$ or later

Communication delay scheduling

ILP formulation

$$
\begin{aligned}
\min \quad & C \\
\text{s.t.} \quad & \textstyle\sum_{i \in \mathrm{Pred}(j)} x_{ij} \;\geq\; |\mathrm{Pred}(j)| - 1 && \text{for } j = 1, \ldots, n \\
& \textstyle\sum_{i \in \mathrm{Succ}(j)} x_{ji} \;\geq\; |\mathrm{Succ}(j)| - 1 && \text{for } j = 1, \ldots, n \\
& C_i + 1 + x_{ij} \;\leq\; C_j && \text{for } J_i \to J_j \\
& 1 \;\leq\; C_j \;\leq\; C && \text{for } j = 1, \ldots, n \\
& x_{ij} \in \{0, 1\} && \text{for } J_i \to J_j
\end{aligned}
$$

Variables:

- C_j: real variable encodes completion time of J_i
- x_{ij}: 0-1-variable encodes delay of $J_i \to J_j$
- C: real variable encodes makespan of schedule

Questions?

Communication delay scheduling

LP relaxation

$$
\begin{aligned}
\min \quad & C \\
\text{s.t.} \quad & \textstyle\sum_{i \in \mathrm{Pred}(j)} x_{ij} \;\geq\; |\mathrm{Pred}(j)| - 1 && \text{for } j = 1, \ldots, n \\
& \textstyle\sum_{i \in \mathrm{Succ}(j)} x_{ji} \;\geq\; |\mathrm{Succ}(j)| - 1 && \text{for } j = 1, \ldots, n \\
& C_i + 1 + x_{ij} \;\leq\; C_j && \text{for } J_i \to J_j \\
& 1 \;\leq\; C_j \;\leq\; C && \text{for } j = 1, \ldots, n \\
& 0 \leq x_{ij} \leq 1 && \text{for } J_i \to J_j
\end{aligned}
$$

Variables:

- C_j: real variable encodes completion time of J_i
- x_{ij}: real variable encodes relaxed delay of $J_i \to J_j$
- C: real variable encodes makespan of schedule

Communication delay scheduling

Approximation algorithm

1. Compute the optimal LP solution x^*_{ij}, C^*_j, C^*.
2. Round the LP solution to a feasible ILP-solution $\tilde{x}_{ij}$, $\tilde{C}_j$, $\tilde{C}$.

How to round the LP solution

For every precedence constraint $J_i \to J_j$ do:
If $x^*_{ij} < 1/2$ then $\tilde{x}_{ij} = 0$
If $x^*_{ij} \geq 1/2$ then $\tilde{x}_{ij} = 1$

For every job J_j do:
$$\tilde{C}_j \;=\; \max\left\{\tilde{C}_i + 1 + \tilde{x}_{ij} \,:\, J_i \to J_j\right\}$$

For the makespan do:
$$\tilde{C} \;=\; \max\{\tilde{C}_i\}$$

Lemma (feasibility)

The rounded solution $\tilde{x}_{ij}$, $\tilde{C}_j$, $\tilde{C}$ is feasible for the ILP.

Lemma (feasibility)

The rounded solution $\tilde{x}_{ij}$, $\tilde{C}_j$, $\tilde{C}$ is feasible for the ILP.

$$\sum_{i \in \mathrm{Pred}(j)} \tilde{x}_{ij} \;\geq\; |\mathrm{Pred}(j)| - 1 \quad \text{and} \quad \sum_{i \in \mathrm{Succ}(j)} \tilde{x}_{ji} \;\geq\; |\mathrm{Succ}(j)| - 1,$$

Lemma (feasibility)

The rounded solution $\tilde{x}_{ij}$, $\tilde{C}_j$, $\tilde{C}$ is feasible for the ILP.

$$\sum_{i \in \mathrm{Pred}(j)} \tilde{x}_{ij} \geq |\mathrm{Pred}(j)| - 1 \quad \text{and} \quad \sum_{i \in \mathrm{Succ}(j)} \tilde{x}_{ji} \geq |\mathrm{Succ}(j)| - 1,$$

since for at most one $i \in \mathrm{Pred}(j)$, we have $x^*_{ij} < 1/2$

and for at most one $i \in \mathrm{Succ}(j)$, we have $x^*_{ji} < 1/2$.

Lemma (feasibility)

The rounded solution $\tilde{x}_{ij}$, $\tilde{C}_j$, $\tilde{C}$ is feasible for the ILP.

$$\sum_{i \in \text{Pred}(j)} \tilde{x}_{ij} \;\geq\; |\text{Pred}(j)| - 1 \quad \text{and} \quad \sum_{i \in \text{Succ}(j)} \tilde{x}_{ji} \;\geq\; |\text{Succ}(j)| - 1,$$

since for at most one $i \in \text{Pred}(j)$, we have $x^*_{ij} < 1/2$

and for at most one $i \in \text{Succ}(j)$, we have $x^*_{ji} < 1/2$.

Constraints

$$C_i + 1 + x_{ij} \;\leq\; C_j$$

are satisfied by construction.

Questions?

Lemma (guarantee, part 1)

For every constraint $J_i \to J_j$, we have $1 + \tilde{x}_{ij} \leq \frac{4}{3}(1 + x^*_{ij})$.

Proof:

Lemma (guarantee, part 1)

For every constraint $J_i \to J_j$, we have $1 + \tilde{x}_{ij} \leq \frac{4}{3}(1 + x^*_{ij})$.

Proof: trivial if $\tilde{x}_{ij} = 0$;

Lemma (guarantee, part 1)

For every constraint $J_i \to J_j$, we have $1 + \tilde{x}_{ij} \leq \frac{4}{3}(1 + x^*_{ij})$.

Proof: trivial if $\tilde{x}_{ij} = 0$; if $\tilde{x}_{ij} = 1$, use $x^*_{ij} \geq 1/2$.

Lemma (guarantee, part 2)

For every job J_j, we have $\tilde{C}_j \leq \frac{4}{3} C_j^*$.

Proof:

Lemma (guarantee, part 2)

For every job J_j, we have $\tilde{C}_j \leq \frac{4}{3} C_j^*$.

Proof: Without loss of generality we assume that each job starts as early as possible on its assigned machine.

Lemma (guarantee, part 2)

For every job J_j, we have $\tilde{C}_j \leq \frac{4}{3} C_j^*$.

Proof: Without loss of generality we assume that each job starts as early as possible on its assigned machine.
Induction on precedence constraint graph.

Lemma (guarantee, part 2)

For every job J_j, we have $\tilde{C}_j \leq \frac{4}{3} C_j^*$.

Proof: Without loss of generality we assume that each job starts as early as possible on its assigned machine.
Induction on precedence constraint graph.
If $|\text{Pred}(j)| = 0$, $\tilde{C}_j = C_j^* = 1$.
If $|\text{Pred}(j)| > 1$, we have that $\tilde{C}_j$ is by definition

$$\max \left\{ \tilde{C}_i + 1 + \tilde{x}_{ij} : \ J_i \to J_j \right\}$$

Lemma (guarantee, part 2)

For every job J_j, we have $\tilde{C}_j \leq \frac{4}{3} C_j^*$.

Proof: Without loss of generality we assume that each job starts as early as possible on its assigned machine.
Induction on precedence constraint graph.
If $|\text{Pred}(j)| = 0$, $\tilde{C}_j = C_j^* = 1$.
If $|\text{Pred}(j)| > 1$, we have that $\tilde{C}_j$ is by definition

$$\max\left\{\tilde{C}_i + 1 + \tilde{x}_{ij} : J_i \to J_j\right\} \leq \max\left\{\tfrac{4}{3} C_i^* + \tfrac{4}{3}(1 + x_{ij}^*) : J_i \to J_j\right\}.$$

Lemma (guarantee, part 2)

For every job J_j, we have $\tilde{C}_j \leq \frac{4}{3} C_j^*$.

Proof: Without loss of generality we assume that each job starts as early as possible on its assigned machine.
Induction on precedence constraint graph.
If $|\mathrm{Pred}(j)| = 0$, $\tilde{C}_j = C_j^* = 1$.
If $|\mathrm{Pred}(j)| > 1$, we have that $\tilde{C}_j$ is by definition

$$\max\left\{\tilde{C}_i + 1 + \tilde{x}_{ij} : J_i \to J_j\right\} \leq \max\left\{\tfrac{4}{3} C_i^* + \tfrac{4}{3}(1 + x_{ij}^*) : J_i \to J_j\right\}.$$

We obtain:

Lemma (guarantee, part 3)

The makespan satisfies $\tilde{C} \leq \frac{4}{3} C^*$.

Questions?

Theorem

This poly-time approximation algorithm has approximation ratio $4/3$.

Is this bound tight?

Theorem

This poly-time approximation algorithm has approximation ratio $4/3$.

Is this bound tight?

Example

- $3k + 1$ jobs $A_1, \ldots, A_{k+1}$; $B_1, \ldots, B_k$; $C_1, \ldots, C_k$
- Precedence constraints:

$A_i \to B_i$ and $A_i \to C_i$ for $i = 1, \ldots, k$

$B_i \to A_{i+1}$ and $C_i \to A_{i+1}$ for $i = 1, \ldots, k$

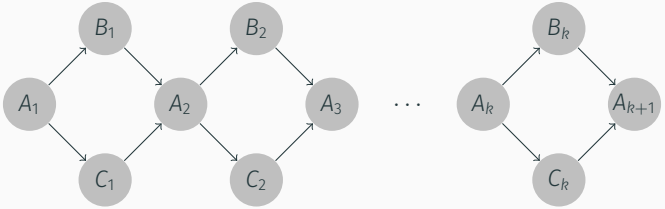
$\to$ What is the minimum makespan for this example?

$\to$ Write down the LP.

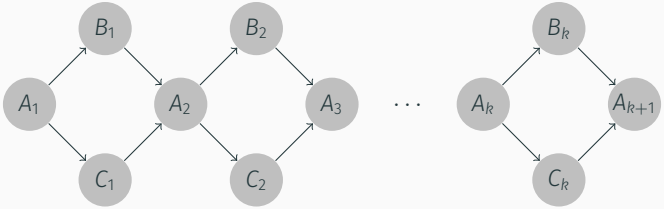
$\to$ What solution will the LP return?

$\to$ What will the approximation algorithm return?

Structure of precedence constraints of the example instances



Structure of precedence constraints of the example instances



What could be good LP-relaxations for this class of instances?

LP in the example

Feasible LP solution:

$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = \frac{1}{2}$ for $i = 1, \ldots k$

LP in the example

Feasible LP solution:
$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = \frac{1}{2}$ for $i = 1, \ldots k$

$$\begin{aligned} \textbf{min} \quad & C \\ \text{s.t.} \quad & \end{aligned}$$

LP in the example

Feasible LP solution:
$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = \frac{1}{2}$ for $i = 1, \ldots k$

$$
\begin{array}{lll}
\textbf{min} & C & \\
\text{s.t.} & \textbf{(Predecessor constraints)} & \\
 & x_{A_iB_i} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{A_iC_i} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{B_iA_{i+1}} + x_{C_iA_{i+1}} \geq 1 & \text{for } i = 1, \ldots, k
\end{array}
$$

LP in the example

Feasible LP solution:
$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = \frac{1}{2}$ for $i = 1, \ldots k$

$$
\begin{array}{lll}
\text{min} & C & \\
\text{s.t.} & \textbf{(Predecessor constraints)} & \\
 & x_{A_iB_i} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{A_iC_i} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{B_iA_{i+1}} + x_{C_iA_{i+1}} \geq 1 & \text{for } i = 1, \ldots, k \\
 & \textbf{(Successor constraints)} & \\
 & x_{B_iA_{i+1}} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{C_iA_{i+1}} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{A_iB_i} + x_{A_iC_i} \geq 1 & \text{for } i = 1, \ldots, k
\end{array}
$$

LP in the example

Feasible LP solution:
$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = \frac{1}{2}$ for $i = 1, \ldots k$

$$
\begin{array}{lll}
\textbf{min} & C & \\
\text{s.t.} & \textbf{(Predecessor constraints)} & \\
 & x_{A_iB_i} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{A_iC_i} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{B_iA_{i+1}} + x_{C_iA_{i+1}} \geq 1 & \text{for } i = 1, \ldots, k \\
 & \textbf{(Successor constraints)} & \\
 & x_{B_iA_{i+1}} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{C_iA_{i+1}} \geq 0 & \text{for } i = 1, \ldots, k \\
 & x_{A_iB_i} + x_{A_iC_i} \geq 1 & \text{for } i = 1, \ldots, k \\
 & \textbf{(Scheduling constraints)} & \\
 & C_{A_i} + 1 + x_{A_iB_i} \leq C_{B_i} & \text{for i=1,...k} \\
 & C_{A_i} + 1 + x_{A_iC_i} \leq C_{C_i} & \text{for i=1,...k} \\
 & C_{B_i} + 1 + x_{B_iA_{i+1}} \leq C_{A_{i+1}} & \text{for i=1,...k} \\
 & C_{C_i} + 1 + x_{C_iA_{i+1}} \leq C_{A_{i+1}} & \text{for i=1,...k}
\end{array}
$$

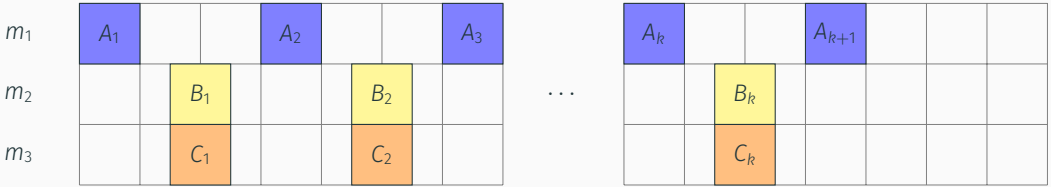
LP in the example

Feasible LP solution:
$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = \frac{1}{2}$ for $i = 1, \ldots k$

min	C	
s.t.	(Predecessor constraints)	
	$x_{A_iB_i} \geq 0$	for $i = 1, \ldots, k$
	$x_{A_iC_i} \geq 0$	for $i = 1, \ldots, k$
	$x_{B_iA_{i+1}} + x_{C_iA_{i+1}} \geq 1$	for $i = 1, \ldots, k$
	(Successor constraints)	
	$x_{B_iA_{i+1}} \geq 0$	for $i = 1, \ldots, k$
	$x_{C_iA_{i+1}} \geq 0$	for $i = 1, \ldots, k$
	$x_{A_iB_i} + x_{A_iC_i} \geq 1$	for $i = 1, \ldots, k$
	(Scheduling constraints)	
	$C_{A_i} + 1 + x_{A_iB_i} \leq C_{B_i}$	for i=1,...k
	$C_{A_i} + 1 + x_{A_iC_i} \leq C_{C_i}$	for i=1,...k
	$C_{B_i} + 1 + x_{B_iA_{i+1}} \leq C_{A_{i+1}}$	for i=1,...k
	$C_{C_i} + 1 + x_{C_iA_{i+1}} \leq C_{A_{i+1}}$	for i=1,...k
	$1 \leq C_j \leq C$	for $j \in \{A_i, B_i, C_i : i = 1, \ldots, k\} \cup \{A_{k+1}\}$
	$x_{j,m} \in [0, 1]$	for $J_j \to J_m$

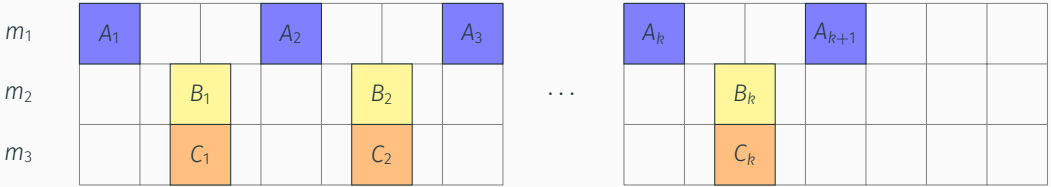
Communication delay scheduling

With all delays at $\frac{1}{2}$, the relaxed schedule looks as follows:



Communication delay scheduling

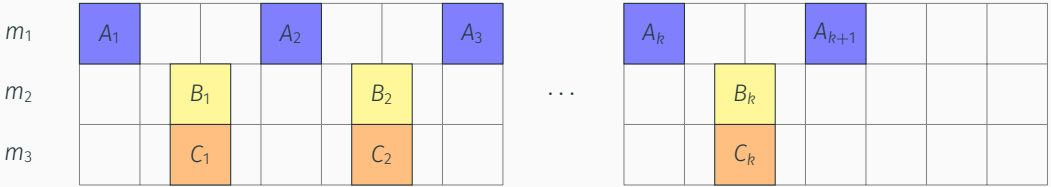
With all delays at $\frac{1}{2}$, the relaxed schedule looks as follows:



$C_{A_1} = 1$, $C_{B_1} = 2.5$, $C_{C_1} = 2.5$, $C_{A_2} = 4$, $C_{B_2} = 5.5$, $C_{C_2} = 5.5$, $C_{A_3} = 7, \ldots$

Communication delay scheduling

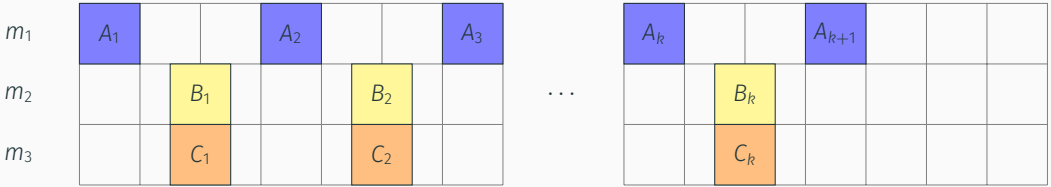
With all delays at $\frac{1}{2}$, the relaxed schedule looks as follows:



$C_{A_1} = 1$, $C_{B_1} = 2.5$, $C_{C_1} = 2.5$, $C_{A_2} = 4$, $C_{B_2} = 5.5$, $C_{C_2} = 5.5$, $C_{A_3} = 7, \ldots$
$C_{A_{k+1}} = 1 + 3k$

Communication delay scheduling

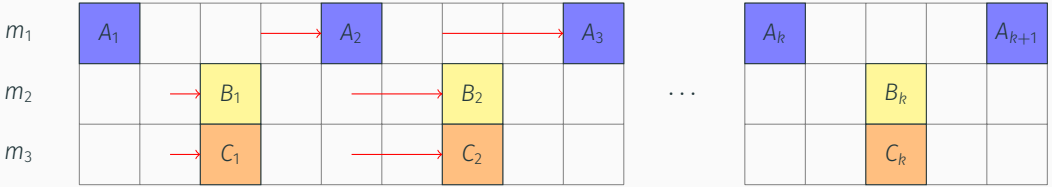
With all delays at $\frac{1}{2}$, the relaxed schedule looks as follows:



$C_{A_1} = 1$, $C_{B_1} = 2.5$, $C_{C_1} = 2.5$, $C_{A_2} = 4$, $C_{B_2} = 5.5$, $C_{C_2} = 5.5$, $C_{A_3} = 7, \ldots$
$C_{A_{k+1}} = 1 + 3k$
Thus the makespan is $1 + 3k$

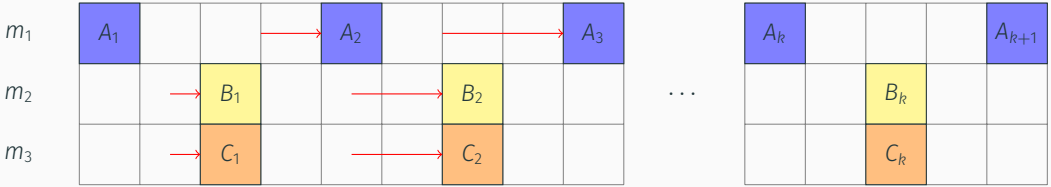
Communication delay scheduling

When rounded, the approximation algorithm obtains the following solution:



Communication delay scheduling

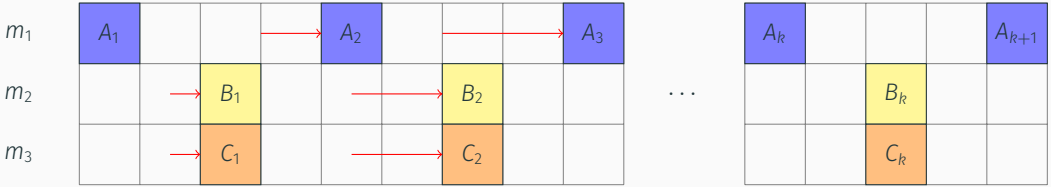
When rounded, the approximation algorithm obtains the following solution:



$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = 1$ for $i = 1, \dots k$

Communication delay scheduling

When rounded, the approximation algorithm obtains the following solution:

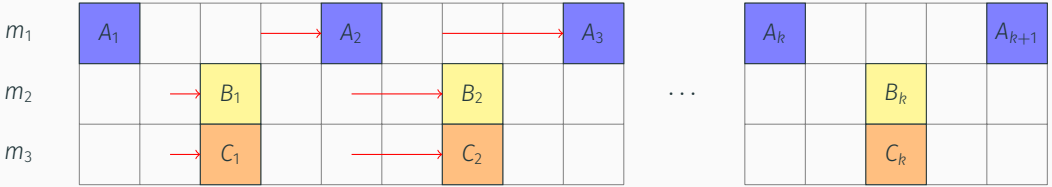


$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = 1$ for $i = 1, \dots k$

$C_{A_1} = 1$, $C_{B_1} = 3$, $C_{C_1} = 3$, $C_{A_2} = 5$, $C_{B_2} = 7$, $C_{C_2} = 7$, $C_{A_3} = 9, \dots$

Communication delay scheduling

When rounded, the approximation algorithm obtains the following solution:



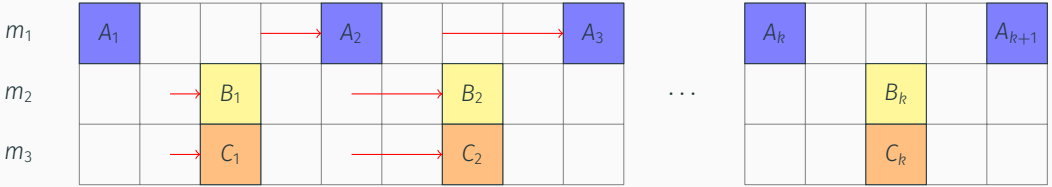
$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = 1$ for $i = 1, \dots k$

$C_{A_1} = 1$, $C_{B_1} = 3$, $C_{C_1} = 3$, $C_{A_2} = 5$, $C_{B_2} = 7$, $C_{C_2} = 7$, $C_{A_3} = 9, \dots$

$C_{A_{k+1}} = 1 + 4k$

Communication delay scheduling

When rounded, the approximation algorithm obtains the following solution:



$x_{A_iB_i} = x_{A_iC_i} = x_{B_iA_{i+1}} = x_{C_iA_{i+1}} = 1$ for $i = 1, \dots k$

$C_{A_1} = 1$, $C_{B_1} = 3$, $C_{C_1} = 3$, $C_{A_2} = 5$, $C_{B_2} = 7$, $C_{C_2} = 7$, $C_{A_3} = 9, \dots$

$C_{A_{k+1}} = 1 + 4k$

Thus the makespan for the approximation algorithm is $1 + 4k$

Communication delay scheduling

A makespan of $1 + 3k$ can be obtained with the following solution:

Communication delay scheduling

A makespan of $1 + 3k$ can be obtained with the following solution:

This solution is optimal, since after each A_i, either B_i or C_i has to incur a delay of 1 (they can not be executed at the same time on the same machine as A_i).

Communication delay scheduling

A makespan of $1 + 3k$ can be obtained with the following solution:

This solution is optimal, since after each A_i, either B_i or C_i has to incur a delay of 1 (they can not be executed at the same time on the same machine as A_i).

If we take an instance I_k from this class of instances, we have

$$\frac{\text{app}(I_k)}{\text{opt}(I_k)} = \frac{1 + 4k}{1 + 3k}$$

Communication delay scheduling

A makespan of $1 + 3k$ can be obtained with the following solution:

This solution is optimal, since after each A_i, either B_i or C_i has to incur a delay of 1 (they can not be executed at the same time on the same machine as A_i).

If we take an instance I_k from this class of instances, we have

$$\frac{\text{app}(I_k)}{\text{opt}(I_k)} = \frac{1 + 4k}{1 + 3k}$$

If we let $k \to \infty$, we obtain

$$\lim_{k \to \infty} \frac{\text{opt}(I_k)}{\text{app}(I_k)} = \frac{4}{3}$$

Communication delay scheduling

A makespan of $1 + 3k$ can be obtained with the following solution:

This solution is optimal, since after each A_i, either B_i or C_i has to incur a delay of 1 (they can not be executed at the same time on the same machine as A_i).

If we take an instance I_k from this class of instances, we have

$$\frac{\text{app}(I_k)}{\text{opt}(I_k)} = \frac{1 + 4k}{1 + 3k}$$

If we let $k \to \infty$, we obtain

$$\lim_{k \to \infty} \frac{\text{opt}(I_k)}{\text{app}(I_k)} = \frac{4}{3}$$

hence we conclude the ratio is tight for out approximation algorithm. ✓

Questions?

Approximation Schemes

Approximation Schemes

Definition (for minimization problem)

A Polynomial Time Approximation Scheme (PTAS) is
a family of approximation algorithms A_ε for $\varepsilon > 0$
with approximation guarantee $1 + \varepsilon$, and
for every fixed ε running time polynomially bounded in instance size

Definition (for minimization problem)

A Polynomial Time Approximation Scheme (PTAS) is
a family of approximation algorithms A_ε for $\varepsilon > 0$
with approximation guarantee $1 + \varepsilon$, and
for every fixed ε running time polynomially bounded in instance size

Typical running times for PTAS:
$n^{1/\varepsilon}$, n^{2/ε^3}, $(1/\varepsilon)^{1/\varepsilon} n^4$, n^2/ε^5, $3^{1/\varepsilon} n^3$, $(4/\varepsilon)!\, n^{2/\varepsilon}$

Definition (for minimization problem)

A Polynomial Time Approximation Scheme (PTAS) is
a family of approximation algorithms A_ε for $\varepsilon > 0$
with approximation guarantee $1 + \varepsilon$, and
for every fixed ε running time polynomially bounded in instance size

Typical running times for PTAS:
$n^{1/\varepsilon}$, n^{2/ε^3}, $(1/\varepsilon)^{1/\varepsilon} n^4$, n^2/ε^5, $3^{1/\varepsilon} n^3$, $(4/\varepsilon)!\, n^{2/\varepsilon}$

For maximization problems
approximation guarantee of A_ε is $1 - \varepsilon$

Questions?

Makespan minimization (1)

Makespan minimization on $m = 2$ machines

Instance: n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to two machines so that the makespan is minimized

Makespan minimization (1)

Makespan minimization on $m = 2$ machines

Instance: n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to two machines so that the makespan is minimized

- Let $L := \max\left\{\max p_i, \quad \frac{1}{2}\sum_{i=1}^{n} p_i\right\}$, and recall $L \leq \text{opt}(I)$

Makespan minimization (1)

Makespan minimization on $m = 2$ machines

Instance: n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to two machines so that the makespan is minimized

- Let $L := \max\left\{\max p_i, \quad \frac{1}{2}\sum_{i=1}^{n} p_i\right\}$, and recall $L \leq \text{opt}(I)$
- Let $\varepsilon > 0$ be desired precision (for worst case ratio $1 + \varepsilon$)

Makespan minimization (1)

Makespan minimization on $m = 2$ machines

Instance: n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to two machines so that the makespan is minimized

- Let $L := \max\left\{\max p_i, \quad \frac{1}{2}\sum_{i=1}^{n} p_i\right\}$, and recall $L \leq \mathrm{opt}(I)$
- Let $\varepsilon > 0$ be desired precision (for worst case ratio $1 + \varepsilon$)

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Makespan minimization (2)

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Makespan minimization (2)

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Analysis of the algorithm:

Makespan minimization (2)

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Analysis of the algorithm:

- running time?
- approximation guarantee?

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:

Step 1 & 4:

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:

Step 1 & 4: $O(n)$

Step 2 & 3:

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$
Step 2 & 3:

- number of big jobs:

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$
Step 2 & 3:

- number of big jobs: $\leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon L}$

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$
Step 2 & 3:

- number of big jobs: $\leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon L} \leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon \frac{1}{2} \sum_{i=1}^{n} p_i}$

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$
Step 2 & 3:

- number of big jobs: $\leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon L} \leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon \frac{1}{2} \sum_{i=1}^{n} p_i} \leq 2/\varepsilon$

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$
Step 2 & 3:

- number of big jobs: $\leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon L} \leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon \frac{1}{2} \sum_{i=1}^{n} p_i} \leq 2/\varepsilon$
- number of assignments of big jobs on machine (disregarding order): $2^{2/\varepsilon}$

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Running time:
Step 1 & 4: $O(n)$
Step 2 & 3:

- number of big jobs: $\leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon L} \leq \frac{\sum_{i=1}^{n} p_i}{\varepsilon \frac{1}{2} \sum_{i=1}^{n} p_i} \leq 2/\varepsilon$
- number of assignments of big jobs on machine (disregarding order): $2^{2/\varepsilon}$
- step 2& 3 in $O(2^{2/\varepsilon} \cdot n)$

Questions?

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with
the assignment of big jobs in optimal schedule

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with
the assignment of big jobs in optimal schedule
- Let B denote the makespan (of big jobs) in that assignment

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

If Greedy increases B: difference in workload between our schedule and optimal schedule

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

If Greedy increases B: difference in workload between our schedule and optimal schedule $\leq \varepsilon L$

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

If Greedy increases B: difference in workload between our schedule and optimal schedule $\leq \varepsilon L$

$\frac{A_\varepsilon(I)}{opt(I)}$

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

If Greedy increases B: difference in workload between our schedule and optimal schedule $\leq \varepsilon L$

$\frac{A_\varepsilon(I)}{opt(I)} \leq \frac{opt(I)+\varepsilon L}{opt(I)}$

Makespan minimization

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

If Greedy increases B: difference in workload between our schedule and optimal schedule $\leq \varepsilon L$

$$\frac{A_\varepsilon(I)}{opt(I)} \leq \frac{opt(I)+\varepsilon L}{opt(I)} \leq \frac{opt(I)+\varepsilon opt(I)}{opt(I)}$$

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \leq \varepsilon L$)
2. Compute all assignments of big jobs to machines
3. For each such assignment,
   add the small jobs greedily to the schedule for big jobs
4. Output the best schedule found

Approximation ratio:

- One of the $2^{2/\varepsilon}$ assignments agrees with

the assignment of big jobs in optimal schedule

- Let B denote the makespan (of big jobs) in that assignment
- If Greedy does not increase B: optimal schedule found

If Greedy increases B: difference in workload between our schedule and optimal schedule $\leq \varepsilon L$

$$\frac{A_\varepsilon(I)}{opt(I)} \leq \frac{opt(I)+\varepsilon L}{opt(I)} \leq \frac{opt(I)+\varepsilon opt(I)}{opt(I)} = 1+\varepsilon$$

Theorem

Makespan minimization on $m = 2$ machines has a PTAS.
More precisely, for any $\varepsilon \leq 1$, a $(1+\epsilon)$-approximation can be found in time $O(2^{2/\varepsilon} \cdot n)$.

Questions?

Definition (for minimization problem)

A Fully Polynomial Time Approximation Scheme (FPTAS) is
a family of approximation algorithms A_ε for $\varepsilon > 0$
with approximation guarantee $1 + \varepsilon$, and
running time polynomially bounded in instance size **and** $\frac{1}{\varepsilon}$

For maximization problems
approximation guarantee of A_ε is $1 - \varepsilon$

Running time examples: n^2/ε, $nm\varepsilon^2$, $(n^2 + m \log m)\varepsilon^{1.5}$, n^3/ε^2, ...

Questions?

In-approximability

Summary: approximation algorithms

- 2-approximation for (weighted) vertex cover
- $\frac{3}{2}$-approximation for metric TSP and visit-each-city-possibly-several-times version in non-metric instances
- $\frac{4}{3}$-approximation for communication delay scheduling
- $(1+\varepsilon)$-approximation for makespan minimization

Summary: approximation algorithms

- 2-approximation for (weighted) vertex cover
- $\frac{3}{2}$-approximation for metric TSP and visit-each-city-possibly-several-times version in non-metric instances
- $\frac{4}{3}$-approximation for communication delay scheduling
- $(1+\varepsilon)$-approximation for makespan minimization

Are these the best polynomial-time approximation algorithms for these problems that are possible?

How do we prove such a statement?

$\rightarrow$ **Inapproximability**

Chromatic number (COLORING)

Instance: an undirected graph $G = (V, E)$

Goal: find proper coloring of V with smallest possible number of colors

(colors $1, 2, \ldots, k$; adjacent vertices receive different colors)

Chromatic number (COLORING)

Instance: an undirected graph $G = (V, E)$
Goal: find proper coloring of V with smallest possible number of colors
(colors $1, 2, \ldots, k$; adjacent vertices receive different colors)

Chromatic number $\chi(G)$ = minimum number of colors in proper coloring

Chromatic number (COLORING)

Instance: an undirected graph $G = (V, E)$
Goal: find proper coloring of V with smallest possible number of colors
(colors $1, 2, \ldots, k$; adjacent vertices receive different colors)

Chromatic number $\chi(G)$ = minimum number of colors in proper coloring

Fact

There exists polynomial time transformation[1] from 3-SAT to COLORING
such that
satisfiable 3-SAT instances translate into graphs with $\chi(G) \leq 3$
unsatisfiable 3-SAT instances translate into graphs with $\chi(G) \geq 4$

[1] See, e.g., `https://www.youtube.com/watch?v=DQdR40emsAc`, or `https://www.youtube.com/watch?v=B5_m8lKULlI` (nicer video, but reduction is from Circuit SAT)

In-approximability

Chromatic number (COLORING)

Instance: an undirected graph $G = (V, E)$
Goal: find proper coloring of V with smallest possible number of colors
(colors $1, 2, \ldots, k$; adjacent vertices receive different colors)

Chromatic number $\chi(G)$ = minimum number of colors in proper coloring

Fact

There exists polynomial time transformation[1] from 3-SAT to COLORING
such that
satisfiable 3-SAT instances translate into graphs with $\chi(G) \leq 3$
unsatisfiable 3-SAT instances translate into graphs with $\chi(G) \geq 4$

Theorem

If COLORING has poly-time approximation algorithm with ratio $r < 4/3$, then P=NP.

[1] See, e.g., `https://www.youtube.com/watch?v=DQdR40emsAc`, or `https://www.youtube.com/watch?v=B5_m8lKULlI` (nicer video, but reduction is from Circuit SAT)

In-approximability

The Gap Technique is a method for establishing in-approximability of a minimization problem X with integral objective values:

1. Take an NP-hard problem Y

2. Construct a poly-time transformation from Y to X
such that
YES-instances of Y translate into X-instances with value $\leq A$
NO-instances of Y translate into X-instances with value $\geq B$

3. Conclude:
If X has poly-time approximation algorithm with ratio $r < B/A$
then P=NP

Questions?

Communication delay scheduling (COMM-DELAY)

Instance: unit time jobs $J_1, \ldots, J_n$;
precedence constraints between some jobs
Goal: find a feasible schedule on n machines
that obeys unit communication delays and minimizes makespan

Communication delay scheduling (COMM-DELAY)

Instance: unit time jobs $J_1, \ldots, J_n$;
precedence constraints between some jobs
Goal: find a feasible schedule on n machines
that obeys unit communication delays and minimizes makespan

Fact (Hoogeveen, Lenstra & Veltman, 1994)

There exists poly-time transformation from 3-SAT to COMM-DELAY
such that
satisfiable 3-SAT instances translate into instances with $\text{opt}(I) \leq 6$
unsatisfiable 3-SAT instances translate into instances with $\text{opt}(I) \geq 7$

In-approximability

Communication delay scheduling (COMM-DELAY)

Instance: unit time jobs $J_1, \ldots, J_n$;
precedence constraints between some jobs
Goal: find a feasible schedule on n machines
that obeys unit communication delays and minimizes makespan

Fact (Hoogeveen, Lenstra & Veltman, 1994)

There exists poly-time transformation from 3-SAT to COMM-DELAY
such that
satisfiable 3-SAT instances translate into instances with $\text{opt}(I) \leq 6$
unsatisfiable 3-SAT instances translate into instances with $\text{opt}(I) \geq 7$

Does this result give us an inapproximability result for COMM-DELAY?

In-approximability

Communication delay scheduling (COMM-DELAY)

Instance: unit time jobs $J_1, \ldots, J_n$;
precedence constraints between some jobs
Goal: find a feasible schedule on n machines
that obeys unit communication delays and minimizes makespan

Fact (Hoogeveen, Lenstra & Veltman, 1994)

There exists poly-time transformation from 3-SAT to COMM-DELAY
such that
satisfiable 3-SAT instances translate into instances with $\text{opt}(I) \leq 6$
unsatisfiable 3-SAT instances translate into instances with $\text{opt}(I) \geq 7$

Theorem

If COMM-DELAY has poly-time approximation algo with ratio $r < 7/6$,
then P=NP.

In-approximability

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof:

In-approximability

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP.

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP. Then the following polynomial-time algorithm solves HC:

In-approximability

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP. Then the following polynomial-time algorithm solves HC:

- Transform an instance I defined by a graph $G = (V, E)$ of HC into an instance I' of TSP by defining distances
$$d(i, j) := \begin{cases} 1 & \text{if } \{i, j\} \in E \\ r \cdot |V| & \text{otherwise} \end{cases}$$

In-approximability

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP. Then the following polynomial-time algorithm solves HC:

- Transform an instance I defined by a graph $G = (V, E)$ of HC into an instance I' of TSP by defining distances
  $$d(i, j) := \begin{cases} 1 & \text{if } \{i, j\} \in E \\ r \cdot |V| & \text{otherwise} \end{cases}$$
- Solve TSP in that graph using A

In-approximability

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP. Then the following polynomial-time algorithm solves HC:

- Transform an instance I defined by a graph $G = (V, E)$ of HC into an instance I' of TSP by defining distances $d(i, j) := \begin{cases} 1 & \text{if } \{i, j\} \in E \\ r \cdot |V| & \text{otherwise} \end{cases}$
- Solve TSP in that graph using A
- If there is a Hamiltonian circuit, then $\text{opt}_{TSP}(I') = |V| \Rightarrow A(I') < r|V|$

In-approximability

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP. Then the following polynomial-time algorithm solves HC:

- Transform an instance I defined by a graph $G = (V, E)$ of HC into an instance I' of TSP by defining distances
  $$d(i, j) := \begin{cases} 1 & \text{if } \{i, j\} \in E \\ r \cdot |V| & \text{otherwise} \end{cases}$$
- Solve TSP in that graph using A
- If there is a Hamiltonian circuit, then $\text{opt}_{TSP}(I') = |V| \Rightarrow A(I') < r|V|$
  If there is no Hamiltonian circuit, then $\text{opt}_{TSP}(I') \geq r|V| \Rightarrow A(I') \geq r|V|$

TSP (Optimization version, visit-each-city-at-most-once version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length **that visits each city at most once**

Theorem

If TSP has poly-time approximation algo with ratio $r < \infty$,
then P=NP.

Proof: Assume there is a polynomial-time approximation algorithm A with approximation ratio $r < \infty$ for the TSP. Then the following polynomial-time algorithm solves HC:

- Transform an instance I defined by a graph $G = (V, E)$ of HC into an instance I' of TSP by defining distances
  $$d(i, j) := \begin{cases} 1 & \text{if } \{i, j\} \in E \\ r \cdot |V| & \text{otherwise} \end{cases}$$
- Solve TSP in that graph using A
- If there is a Hamiltonian circuit, then $\text{opt}_{TSP}(I') = |V| \Rightarrow A(I') < r|V|$
  If there is no Hamiltonian circuit, then $\text{opt}_{TSP}(I') \geq r|V| \Rightarrow A(I') \geq r|V|$
- If $A(I') < r|V|$, return 'YES', if $A(I') \geq r|V|$, return 'NO'.

Questions?

Other examples of approximation schemes

Other examples of approximation schemes: Euclidean TSP

Euclidean TSP

Instance: Points $(x_1, y_1), \ldots, (x_n, y_n)$ in the plane $\mathbb{R}^2$ in the plane
Goal: Find a Round-tour that minimizes the total Euclidean distance

Theorem (Arora, Mitchell)

There is an $(1+\varepsilon)$-approximation algorithm for Euclidean TSP running in time $O(n^{O(\varepsilon)})$.

- The running time was later improved to $O(n(\log n)^{1/\varepsilon})$ by Arora.
- Result was found independently by Arora and Mitchell, both received the Gödel prize for it.
- We'll skip a detailed exposition of this algorithm in this course.

Other examples of approximation schemes: Knapsack

Knapsack

Instance: Items $1, \ldots, n$, each with a weight w_i and value v_i; an integer W.
Goal: Find a subset $X \subseteq \{1, \ldots, n\}$ maximizing $\sum_{i \in X} v_i$ under the constraint that $\sum_{i \in X} w_i \leq W$.

Knapsack is weakly NP-Complete.

Theorem (Folklore)

There is an algorithm $(1 - \varepsilon)$-approximation algorithm for Knapsack running in time $O(n^3/\varepsilon)$.

- The algorithm is discussed in week 7.

Other examples of approximation schemes: Planar Independent Set

Planar graph: A graph that admit a drawing in $\mathbb{R}^2$ without crossing of edges.

Planar Independent Set

Instance: A planar graph G.
Goal: Find a independent of G of maximum size.

Planar independent set is NP-complete.

Theorem (Baker)

There is an algorithm $(1-\varepsilon)$*-approximation algorithm for Planar Independent Set running in time* $O(2^{O(1/\varepsilon)}n^4)$.

- The algorithm is discussed in week 8.

Questions?

Following weeks

From next week, Tom van der Zanden takes over the AC course.
He will contact you via email. There will be a new zoom link.

Thank you! Goodbye! Enjoy the rest of the course!