Algorithms and Complexity (AC), week 5

Marie Schmidt

based on slides by Jesper Nederlof

LNMB, Sep–Nov 2019

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

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 (COMM-DELAY)

Instance: unit time jobs $J_1, \dots, J_n$; 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 $J_1, \dots, J_n$; 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 $C(J_a) := S(J_a) + 1$

Communication delay scheduling (COMM-DELAY)

Instance: unit time jobs $J_1, \dots, J_n$; 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

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 $C(J_a) \leq S(J_b)$

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

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 $J_a \rightarrow J_b$

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 "$\to$" on the jobs
- if $J_a \to J_b$ then $C(J_a) \leq S(J_b)$
  $\iff$ J_a must be completed before J_b is started
- unit communication delay for $J_a \to J_b$
  if J_a and J_b run on same machine then $C(J_a) \leq S(J_b)$

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 $C(J_a) + 1 \leq S(J_b)$

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 (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2;\ \ J_1 \rightarrow J_3;\ \ J_2 \rightarrow J_4;\ \ J_3 \rightarrow J_4;$

Communication delay scheduling (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2;\ \ J_1 \rightarrow J_3;\ \ J_2 \rightarrow J_4;\ \ J_3 \rightarrow J_4;$

Lower bound:

Communication delay scheduling (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2;\ \ J_1 \rightarrow J_3;\ \ J_2 \rightarrow J_4;\ \ J_3 \rightarrow J_4;$

Lower bound: makespan ≥ 3

Communication delay scheduling (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2$; $J_1 \rightarrow J_3$; $J_2 \rightarrow J_4$; $J_3 \rightarrow J_4$;

Lower bound: makespan ≥ 3

- Simple schedule:
  If all four jobs are run on different machines:

Communication delay scheduling (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2$; $J_1 \rightarrow J_3$; $J_2 \rightarrow J_4$; $J_3 \rightarrow J_4$;

Lower bound: makespan ≥ 3

- Simple schedule:
  If all four jobs are run on different machines: makespan=5

Communication delay scheduling (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2;\ \ J_1 \rightarrow J_3;\ \ J_2 \rightarrow J_4;\ \ J_3 \rightarrow J_4;$

Lower bound: makespan ≥ 3

- Simple schedule:
  If all four jobs are run on different machines: makespan=5
- Better schedule:

Communication delay scheduling (2)

Example

- Four jobs J_1, J_2, J_3, J_4
- Precedence constraints:
  $J_1 \rightarrow J_2;\ \ J_1 \rightarrow J_3;\ \ J_2 \rightarrow J_4;\ \ J_3 \rightarrow J_4;$

Lower bound: makespan ≥ 3

- Simple schedule:
  If all four jobs are run on different machines: makespan=5
- Better schedule:
  If all four jobs are run on same machine then makespan=4

Communication delay scheduling (3a)

Notation:

- Pred(J_a) denotes the set of all predecessors J_b of J_a (with $J_b \rightarrow J_a$)
- Succ(J_a) denotes the set of all successors J_b of J_a (with $J_a \rightarrow J_b$)

Communication delay scheduling (3a)

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 $C(J_a)$.

Communication delay scheduling (3a)

Notation:

- Pred(J_a) denotes the set of all predecessors J_b of J_a (with $J_b \rightarrow J_a$)
- Succ(J_a) denotes the set of all successors J_b of J_a (with $J_a \rightarrow J_b$)

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 $J_a \rightarrow J_b$

- $x_{ab} = 0$ 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 (3a)

Notation:

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

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 $J_a \rightarrow J_b$

- $x_{ab} = 0$ 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

Corresponding inequality: $C(J_b) \geq C(J_a) + 1 + x_{ab}$

Communication delay scheduling (3a)

Notation:

- Pred(J_a) denotes the set of all predecessors J_b of J_a (with $J_b \to J_a$)
- 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

Corresponding inequality: $C(J_b) \geq C(J_a) + 1 + x_{ab}$

Observation

$$C(J_b) \;=\; \max\{C(J_a) + 1 + x_{ab} : \; J_a \to J_b\}$$

Communication delay scheduling (3b)

ILP formulation

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

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

Communication delay scheduling (3c)

LP relaxation

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

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 (4)

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

Communication delay scheduling (4)

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 $\tilde{x}_{ij} = 1$

Communication delay scheduling (4)

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:

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

Communication delay scheduling (4)

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:

$$\tilde{C} \;=\; \max\{\tilde{C}_i\}$$

Communication delay scheduling (5)

Lemma (feasibility)

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

Communication delay scheduling (5)

Lemma (feasibility)

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

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

Communication delay scheduling (5)

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 $x^*_{ij} < 1/2$

Communication delay scheduling (5)

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 $x^*_{ji} < 1/2$.

Communication delay scheduling (5)

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 $x^*_{ji} < 1/2$.

Constraint on C_i's is satisfied by construction.

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

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 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: Induction on precedence constraint graph.

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: Induction on precedence constraint graph.
If $|\mathrm{Pred}(j)| = 0$, $\tilde{C}_j = C^*_j = 1$.

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: 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

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

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: 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

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

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: 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 $\tilde{C} \leq \frac{4}{3} C^*$.

Communication delay scheduling (7)

Theorem

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

Communication delay scheduling (8a): Gaps

Is this bound tight?

Communication delay scheduling (8a): Gaps

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 $i = 1, \ldots, k$

Communication delay scheduling (8b): Integrality Gap

Not discussed

Example

- Job are partitioned into $k+1$ levels $0, 1, \ldots, k$, with 2^i jobs at level i
- Every job at level i has two successors at level $i+1$
  Every job at level i has one predecessor at level $i-1$

- $\text{opt}_{ILP} \geq 2k+1$
- $\text{opt}_{LP} \leq \frac{3}{2}k+1$ $\quad (x^*_{ij} = 1/2$ for all constraints $J_i \rightarrow J_j)$

Observation

For large numbers of jobs, opt_{ILP} may come arbitrarily close to $\frac{4}{3}\text{opt}_{LP}$.

Therefore the integrality gap of our LP relaxation is $4/3$.

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

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

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

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

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 $1 - \varepsilon$

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 $p_1, \dots, p_n$
Goal: assign jobs to two machines so that the makespan is minimized

- Let $L := \max\left\{\max p_i, \ \frac{1}{2}\sum_{i=1}^{n} p_i\right\}$, 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 (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

Running time:

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

Running time:
Step 1 & 4:

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

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

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

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

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

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

- number of big jobs:

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

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

- number of big jobs: $\leq 2/\varepsilon$

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

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

- number of big jobs: $\leq 2/\varepsilon$
- number of assignments of big jobs per machine: $2^{2/\varepsilon}$

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

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

- number of big jobs: $\leq 2/\varepsilon$
- number of assignments of big jobs per machine: $2^{2/\varepsilon}$
- step 3 in $O(2^{2/\varepsilon} \cdot n)$

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:

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

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

Approximation algorithm

1. Classify processing times into big ($p_j > \varepsilon L$) and small ($p_j \le \varepsilon L$)
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 $\leq \varepsilon L$

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

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 $\leq \varepsilon L$

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

Makespan minimization (3)

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 $O(2^{2/\varepsilon} \cdot n)$.

Fully Polynomial Time Approximation Schemes

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 $1 - \varepsilon$

Summary: approximation algorithms

- 2-approximation for (weighted) vertex cover
- $\frac{3}{2}$-approximation for metric TSP
- $\frac{4}{3}$-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
- $\frac{4}{3}$-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**

In-approximability (1)

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)

In-approximability (1)

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

In-approximability (1)

Chromatic number (COLORING)

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

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

Fact

There exists polynomial time transformation 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 $\chi(G) \geq 4$

In-approximability (1)

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 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.

In-approximability (2)

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

In-approximability (2)

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 Is with $\text{opt}(I) \leq 6$
unsatisfiable 3-SAT instances translate into graphs with $\text{opt}(I) \geq 7$

In-approximability (2)

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 Is with $\text{opt}(I) \leq 6$
unsatisfiable 3-SAT instances translate into graphs 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 (3)

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

In-approximability (4)

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i,j)$
Goal: find roundtrip of smallest possible length

Theorem

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

Proof:

In-approximability (4)

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i,j)$
Goal: find roundtrip of smallest possible length

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.

In-approximability (4)

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length

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 (4)

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i,j)$
Goal: find roundtrip of smallest possible length

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 (4)

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i,j)$
Goal: find roundtrip of smallest possible length

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 (4)

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i,j)$
Goal: find roundtrip of smallest possible length

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 $A(I') \leq |V|$, return 'YES', if $A(I') \geq |V|$, return 'NO'.

Other examples of approximation schemes: Euclidean TSP

Euclidean TSP

Instance: Points $(x_1, y_1), \dots, (x_n, y_n)$ in the plane $\mathbb{R}^2$ in the plane
Goal: Find a Round-tour visiting 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 v.
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 $O(n^3/\varepsilon)$.

- We'll see the algorithm 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 $O(2^{O(1/\varepsilon)}n^4)$.

- We'll see the algorithm in week 8.

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

Recommended reading

Cormen, Leiserson, Rivest and Stein ‘Introduction to Algorithms’:

- Chapter 35 (Approximation Algorithms)

Garey, Johnson 'Computers and Intractability'

- Chapter 6 (Coping with NP-complete problems)