Additive Approximation Schemes for Load Balancing problems

Moritz Buchem [1] Lars Rohwedder [2]
Tjark Vredeveld [1] Andreas Wiese [3]

[1]Maastricht University

[2]École Polytechnique Fédérale de Lausanne

[3]Universidad de Chile

March 2021

Load balancing problems on Identical Parallel machines

- **Input:** n jobs with processing time p_j need to be assigned to m identical machines (where m is part of the input)

Load balancing problems on Identical Parallel machines

- **Input:** n jobs with processing time p_j need to be assigned to m identical machines (where m is part of the input)
- **Schedule:** an assignment of jobs to machines, where a machine load is

$$C_i = \sum_{j \to i} p_j$$

Load balancing problems on Identical Parallel machines

- **Input:** n jobs with processing time p_j need to be assigned to m identical machines (where m is part of the input)
- **Schedule:** an assignment of jobs to machines, where a machine load is

$$C_i = \sum_{j \to i} p_j$$

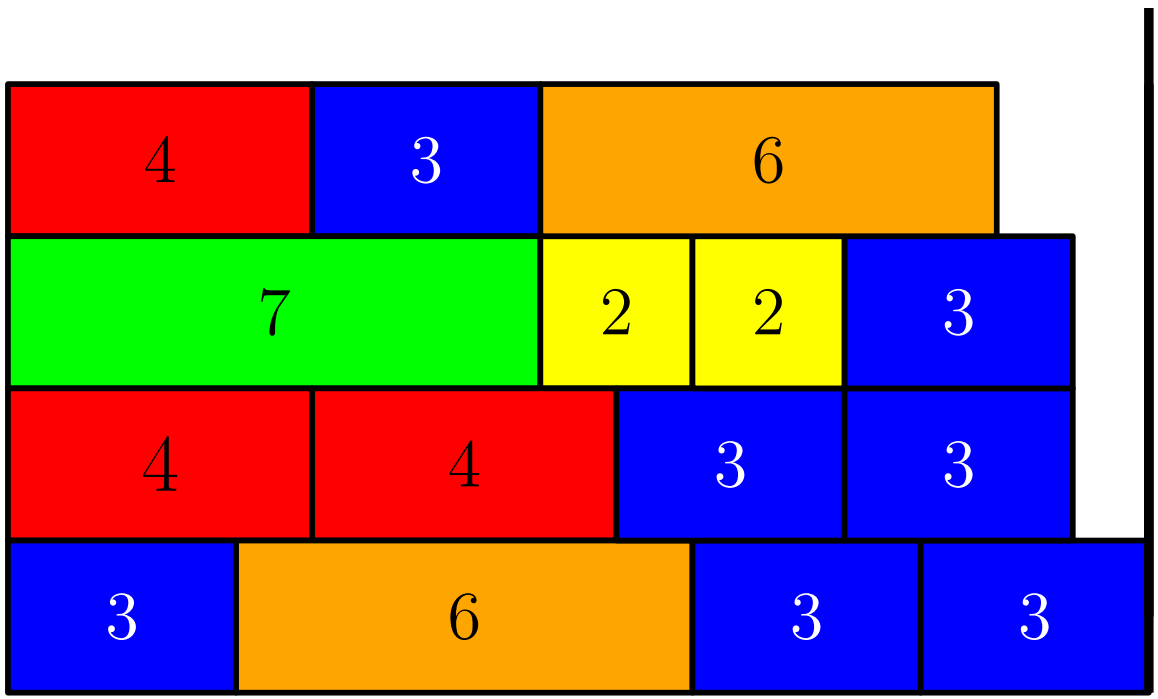
- **Goal:**
  1. *Makespan minimization:* $\min \max_i \{C_i\}$
  2. *Santa Claus problem:* $\max \min_i \{C_i\}$
  3. *Envy-minimizing Santa Claus problem:* $\min \max_i \{C_i\} - \min_i \{C_i\}$

Makespan minimization

Objective:

$$\min \max_i \{C_i\}$$

$$C_{\max} = 15$$

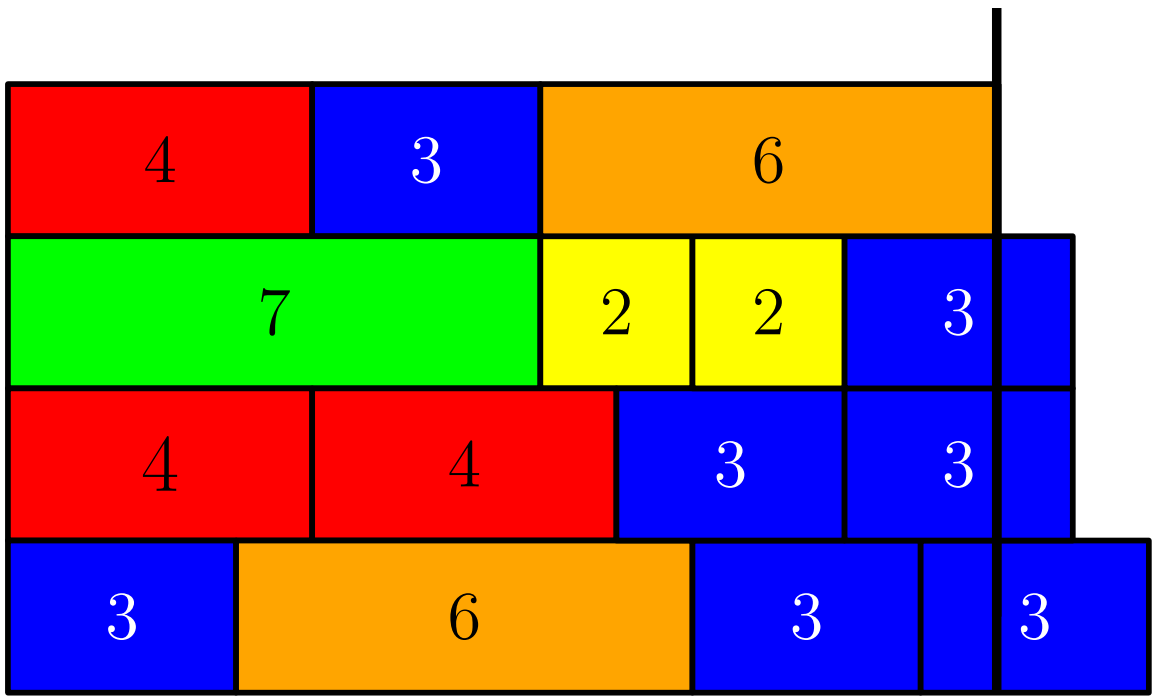


Santa Claus

Objective:

$$\max \min_i \{C_i\}$$

$$C_{\min} = 13$$

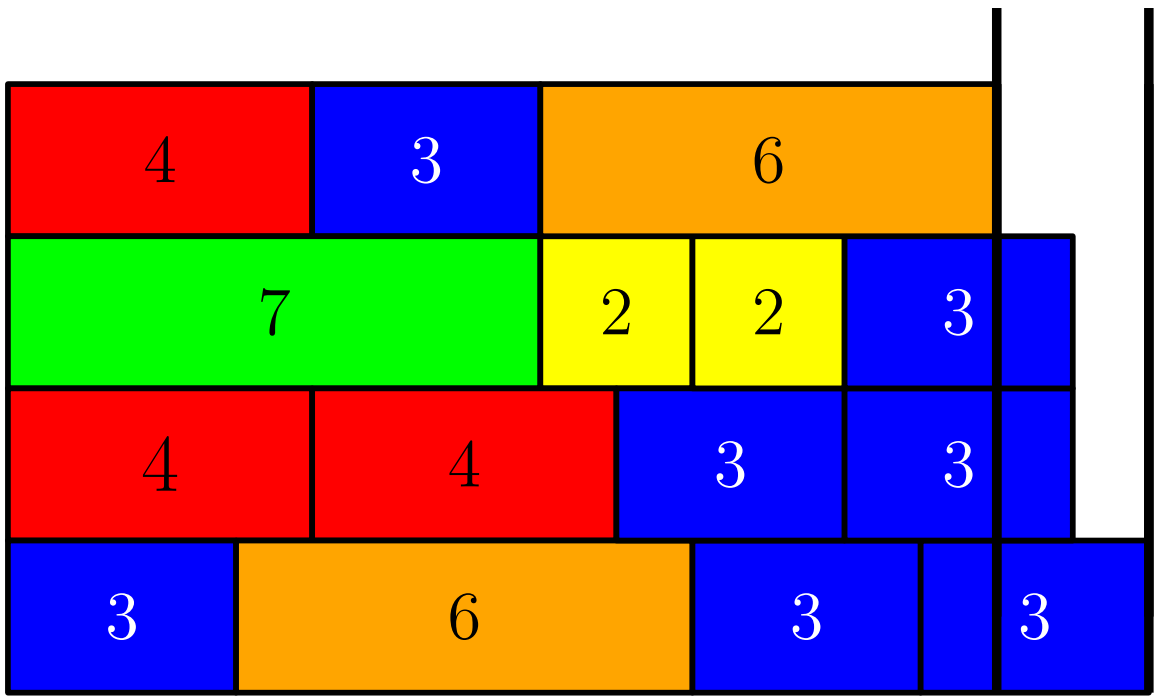


Envy-minimizing Santa Claus

Objective:

$$\min \max_i \{C_i\} - \min_i \{C_i\}$$

$$C_{\max} - C_{\min} = 2$$



Related work

- **Complexity:** Load balancing problems are strongly $\mathcal{NP}$-hard [3] when m is part of the input

Related work

- **Complexity:** Load balancing problems are strongly $\mathcal{NP}$-hard [3] when m is part of the input
- **Approximation results:**

Related work

- **Complexity:** Load balancing problems are strongly $\mathcal{NP}$-hard [3] when m is part of the input
- **Approximation results:**
  - *Makespan minimization and Santa Claus:* (Efficient) polynomial time approximation schemes exist [1, 2, 4, 5, 6, 7]

Related work

- **Complexity:** Load balancing problems are strongly $\mathcal{NP}$-hard [3] when m is part of the input
- **Approximation results:**
  - *Makespan minimization and Santa Claus:* (Efficient) polynomial time approximation schemes exist [1, 2, 4, 5, 6, 7]
  - *Envy-minimizing Santa Claus:* No constant factor approximation possible, unless $P = NP$

Our result

- **Additive approximation schemes:** Given $\epsilon > 0$, can we find solutions with an additive performance guarantee depending on the maximum processing time and ϵ, i.e.

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

Our result

- **Additive approximation schemes:** Given $\epsilon > 0$, can we find solutions with an additive performance guarantee depending on the maximum processing time and ϵ, i.e.

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

  - Stronger guarantee when $p_{\max} \ll \mathrm{OPT}$

Our result

- **Additive approximation schemes:** Given $\epsilon > 0$, can we find solutions with an additive performance guarantee depending on the maximum processing time and ϵ, i.e.

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

  - > Stronger guarantee when $p_{\max} \ll \mathrm{OPT}$
  - > Alternative when no multiplicative approximation is possible

Our result

- **Additive approximation schemes:** Given $\epsilon > 0$, can we find solutions with an additive performance guarantee depending on the maximum processing time and ϵ, i.e.

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

  - Stronger guarantee when $p_{\max} \ll \mathrm{OPT}$
  - Alternative when no multiplicative approximation is possible

Theorem (Additive polynomial time approximation scheme)

For any $\epsilon > 0$, we can find a solution for the makespan minimization, Santa Claus or envy-minimizing Santa Claus problem with a value ALG satisfying

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

in running time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Additive approximation scheme for makespan minimization

1. Introduce **slot-MILP**-relaxation

Additive approximation scheme for makespan minimization

1. Introduce **slot-MILP**-relaxation
2. Derive **structural properties** of optimal solutions to the slot-MILP and use these

Additive approximation scheme for makespan minimization

1. Introduce **slot-MILP**-relaxation
2. Derive **structural properties** of optimal solutions to the slot-MILP and use these
3. Using **local search** to **round** the slot-MILP solution in order to obtain a schedule/assignment

The slot-MILP relaxation

- **Job types:** Partition $\mathcal{J}$ into $\mathcal{J}_1, \ldots, \mathcal{J}_{1/\epsilon}$, where

$$\mathcal{J}_k = \{j \in J | p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]\}.$$

The slot-MILP relaxation

- **Job types:** Partition $\mathcal{J}$ into $\mathcal{J}_1, \ldots, \mathcal{J}_{1/\epsilon}$, where

$$\mathcal{J}_k = \{j \in J | p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]\}.$$

- **Integer slot variables:** Number of slots for jobs of type k on machine i.

The slot-MILP relaxation

- **Job types:** Partition $\mathcal{J}$ into $\mathcal{J}_1, \ldots, \mathcal{J}_{1/\epsilon}$, where

$$\mathcal{J}_k = \{j \in J | p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]\}.$$

- **Integer slot variables:** Number of slots for jobs of type k on machine i.
- **Fractional assignment variables:** Assignment of job j to machine i.

The slot-MILP relaxation

- **Job types:** Partition $\mathcal{J}$ into $\mathcal{J}_1, \ldots, \mathcal{J}_{1/\epsilon}$, where

$$\mathcal{J}_k = \{j \in J | p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]\}.$$

- **Integer slot variables:** Number of slots for jobs of type k on machine i.
- **Fractional assignment variables:** Assignment of job j to machine i.

$$\begin{aligned}
&\min u && \\
&\sum_{i \in \mathcal{M}} x_{i,j} = 1 && \forall j \in \mathcal{J} \\
&\sum_{j \in \mathcal{J}} p_j x_{i,j} \leq u && \forall i \in \mathcal{M} \\
&\sum_{j \in \mathcal{J}_k} x_{i,j} = y_{i,k} && \forall i \in \mathcal{M}, \forall k \in \{1, \ldots, 1/\epsilon\} \\
&x_{i,j} \geq 0 && \forall j \in \mathcal{J}, i \in \mathcal{M} \\
&y_{i,k} \in \mathbb{N}_0 && \forall i \in \mathcal{M}, k \in \{1, \ldots, 1/\epsilon\}
\end{aligned}$$

Solving the slot-MILP: Structural property

- **Negative:** Slot-MILP contains $m \cdot \frac{1}{\epsilon}$ many integer variables so we can not apply results for (M)ILPs of fixed dimension

Solving the slot-MILP: Structural property

- **Negative:** Slot-MILP contains $m \cdot \frac{1}{\epsilon}$ many integer variables so we can not apply results for (M)ILPs of fixed dimension
- **Positive:** We can reduce the number of slot vectors $(y_i = (y_{i,1}, \ldots, y_{i,\frac{1}{\epsilon}}))$ to be considered

Solving the slot-MILP: Structural property

Lemma

There is an optimal solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.

Solving the slot-MILP: Structural property

Lemma

There is an optimal solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.

Proof.

- Let (x, y) be an optimal solution in which for $i_1, i_2 \in \mathcal{M}$ and $y_{i_1} \equiv y_{i_2} \bmod 2$ it follows that $y_{i_1} \neq y_{i_2}$. Suppose that (x, y) minimizes potential function.

□

Solving the slot-MILP: Structural property

Lemma

There is an optimal solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.

Proof.

- Let (x, y) be an optimal solution in which for $i_1, i_2 \in \mathcal{M}$ and $y_{i_1} \equiv y_{i_2} \bmod 2$ it follows that $y_{i_1} \neq y_{i_2}$. Suppose that (x, y) minimizes potential function.
- New solution (x', y'): take the average of machines i_1 and i_2 and leave all other machines unchanged.

□

Solving the slot-MILP: Structural property

Lemma

There is an optimal solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.

Proof.

- Let (x, y) be an optimal solution in which for $i_1, i_2 \in \mathcal{M}$ and $y_{i_1} \equiv y_{i_2} \bmod 2$ it follows that $y_{i_1} \neq y_{i_2}$. Suppose that (x, y) minimizes potential function.
- New solution (x', y'): take the average of machines i_1 and i_2 and leave all other machines unchanged.
  - (x', y') is feasible due to property of (x, y).

□

Solving the slot-MILP: Structural property

Lemma

There is an optimal solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.

Proof.

- Let (x, y) be an optimal solution in which for $i_1, i_2 \in \mathcal{M}$ and $y_{i_1} \equiv y_{i_2} \bmod 2$ it follows that $y_{i_1} \neq y_{i_2}$. Suppose that (x, y) minimizes potential function.
- New solution (x', y'): take the average of machines i_1 and i_2 and leave all other machines unchanged.
  - (x', y') is feasible due to property of (x, y).
  - Average load of i_1 and i_2 satisfies upper bound u

□

Solving the slot-MILP: Structural property

Lemma

There is an optimal solution (x, y) to the slot-MILP such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it follows that $y_i = y_{i'}$.

Proof.

- Let (x, y) be an optimal solution in which for $i_1, i_2 \in \mathcal{M}$ and $y_{i_1} \equiv y_{i_2} \bmod 2$ it follows that $y_{i_1} \neq y_{i_2}$. Suppose that (x, y) minimizes potential function.
- New solution (x', y'): take the average of machines i_1 and i_2 and leave all other machines unchanged.
  - (x', y') is feasible due to property of (x, y).
  - Average load of i_1 and i_2 satisfies upper bound u
  - Potential function is decreased

□

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Proof.

- Total of $2^{\frac{1}{\epsilon}}$ different machine types

□

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Proof.

- Total of $2^{\frac{1}{\epsilon}}$ different machine types
- For each type we guess:

□

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Proof.

- Total of $2^{\frac{1}{\epsilon}}$ different machine types
- For each type we guess:
  1. Number of machines of this type $m^{O\left(2^{1/\epsilon}\right)}$

□

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Proof.

- Total of $2^{\frac{1}{\epsilon}}$ different machine types
- For each type we guess:
  1. Number of machines of this type $m^{O\left(2^{1/\epsilon}\right)}$
  2. for each $k \in \{1, ..., 1/\epsilon\}$ we guess the value of $y_{i,k}$ for each machine $i \in \mathcal{M}$ of this type $n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$

□

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Proof.

- Total of $2^{\frac{1}{\epsilon}}$ different machine types
- For each type we guess:
  1. Number of machines of this type $m^{O\left(2^{1/\epsilon}\right)}$
  2. for each $k \in \{1, ..., 1/\epsilon\}$ we guess the value of $y_{i,k}$ for each machine $i \in \mathcal{M}$ of this type $n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$
- Total guesses: $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$

□

Solving the slot-MILP: Exact solution

Lemma

We can solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Proof.

- Total of $2^{\frac{1}{\epsilon}}$ different machine types
- For each type we guess:
  1. Number of machines of this type $m^{O\left(2^{1/\epsilon}\right)}$
  2. for each $k \in \{1, \ldots, 1/\epsilon\}$ we guess the value of $y_{i,k}$ for each machine $i \in \mathcal{M}$ of this type $n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$
- Total guesses: $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$
- For each guess: Find optimal assignment variables and continue with the guess yielding the smallest u

□

Final step: Rounding the slot-MILP solution

- Solution to slot-MILP gives integer number of slots but fractional assignment of jobs

Final step: Rounding the slot-MILP solution

- Solution to slot-MILP gives integer number of slots but fractional assignment of jobs
- Assignment needs to be rounded without violating the machine load bound by more than $\epsilon p_{\max}$

Final step: Rounding the slot-MILP solution

- Solution to slot-MILP gives integer number of slots but fractional assignment of jobs
- Assignment needs to be rounded without violating the machine load bound by more than $\epsilon p_{\max}$

Local Search!

Rounding via Local Search - Fixing overloaded machines

1 **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

Rounding via Local Search - Fixing overloaded machines

1. **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

2. $\mathcal{M}_1$: Set of overloaded machines

Rounding via Local Search - Fixing overloaded machines

1. **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

2. $\mathcal{M}_1$: Set of overloaded machines
   1. **Swap jobs:**

Rounding via Local Search - Fixing overloaded machines

1. **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

2. $\mathcal{M}_1$: Set of overloaded machines
   1. **Swap jobs:**
      - Find jobs $j, j' \in \mathcal{J}_k$ with $p_{j'} < p_j$ such that j is assigned to $i \in \mathcal{M}_1$ and j' to $i' \notin \mathcal{M}_1$ with a load of at most u.

Rounding via Local Search - Fixing overloaded machines

1. **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

2. $\mathcal{M}_1$: Set of overloaded machines
   1. **Swap jobs:**
      - Find jobs $j, j' \in \mathcal{J}_k$ with $p_{j'} < p_j$ such that j is assigned to $i \in \mathcal{M}_1$ and j' to $i' \notin \mathcal{M}_1$ with a load of at most u.
      - Swap j and j' between i and i'.

Rounding via Local Search - Fixing overloaded machines

1. **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

2. $\mathcal{M}_1$: Set of overloaded machines
   1. **Swap jobs:**
      - Find jobs $j, j' \in \mathcal{J}_k$ with $p_{j'} < p_j$ such that j is assigned to $i \in \mathcal{M}_1$ and j' to $i' \notin \mathcal{M}_1$ with a load of at most u.
      - Swap j and j' between i and i'.
      - If for any such pair the load on machine i' is more than u, let $\mathcal{M}_2$ be the set of these machines.

Rounding via Local Search - Fixing overloaded machines

1. **Initial solution:** Assign $y_{i,k}$ arbitrary jobs of type k to machine i. This may lead to overloaded machines:

$$> u + \epsilon p_{\max}$$

2. $\mathcal{M}_1$: Set of overloaded machines
   1. **Swap jobs:**
      - Find jobs $j, j' \in \mathcal{J}_k$ with $p_{j'} < p_j$ such that j is assigned to $i \in \mathcal{M}_1$ and j' to $i' \notin \mathcal{M}_1$ with a load of at most u.
      - Swap j and j' between i and i'.
      - If for any such pair the load on machine i' is more than u, let $\mathcal{M}_2$ be the set of these machines.
   2. Repeat the procedure until for $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$ we can find a pair of jobs to swap between a machine in $\mathcal{M}_\ell$ and a not labelled machine yet. After that restart with $\mathcal{M}_1$.

Breadth First Search Formulation

- Graph on $n + 1$ nodes:

Breadth First Search Formulation

- Graph on $n + 1$ nodes:
  - **Source node** s

Breadth First Search Formulation

- Graph on $n + 1$ nodes:
  - **Source node** s
  - **Slot node** associated with a slot $y_{i,k}$ and the processing time of the job assigned to this slot

Breadth First Search Formulation

- Graph on $n + 1$ nodes:
  - **Source node** s
  - **Slot node** associated with a slot $y_{i,k}$ and the processing time of the job assigned to this slot
- **Three types of arcs**:

Breadth First Search Formulation

- Graph on $n + 1$ nodes:
  - **Source node** s
  - **Slot node** associated with a slot $y_{i,k}$ and the processing time of the job assigned to this slot
- **Three types of arcs**:
  - Slots on the same machine form a clique with arcs of weight 0

Breadth First Search Formulation

- Graph on $n + 1$ nodes:
  - **Source node** s
  - **Slot node** associated with a slot $y_{i,k}$ and the processing time of the job assigned to this slot
- **Three types of arcs**:
  - Slots on the same machine form a clique with arcs of weight 0
  - An arc from s to every slot on an overloaded machine of weight 1

Breadth First Search Formulation

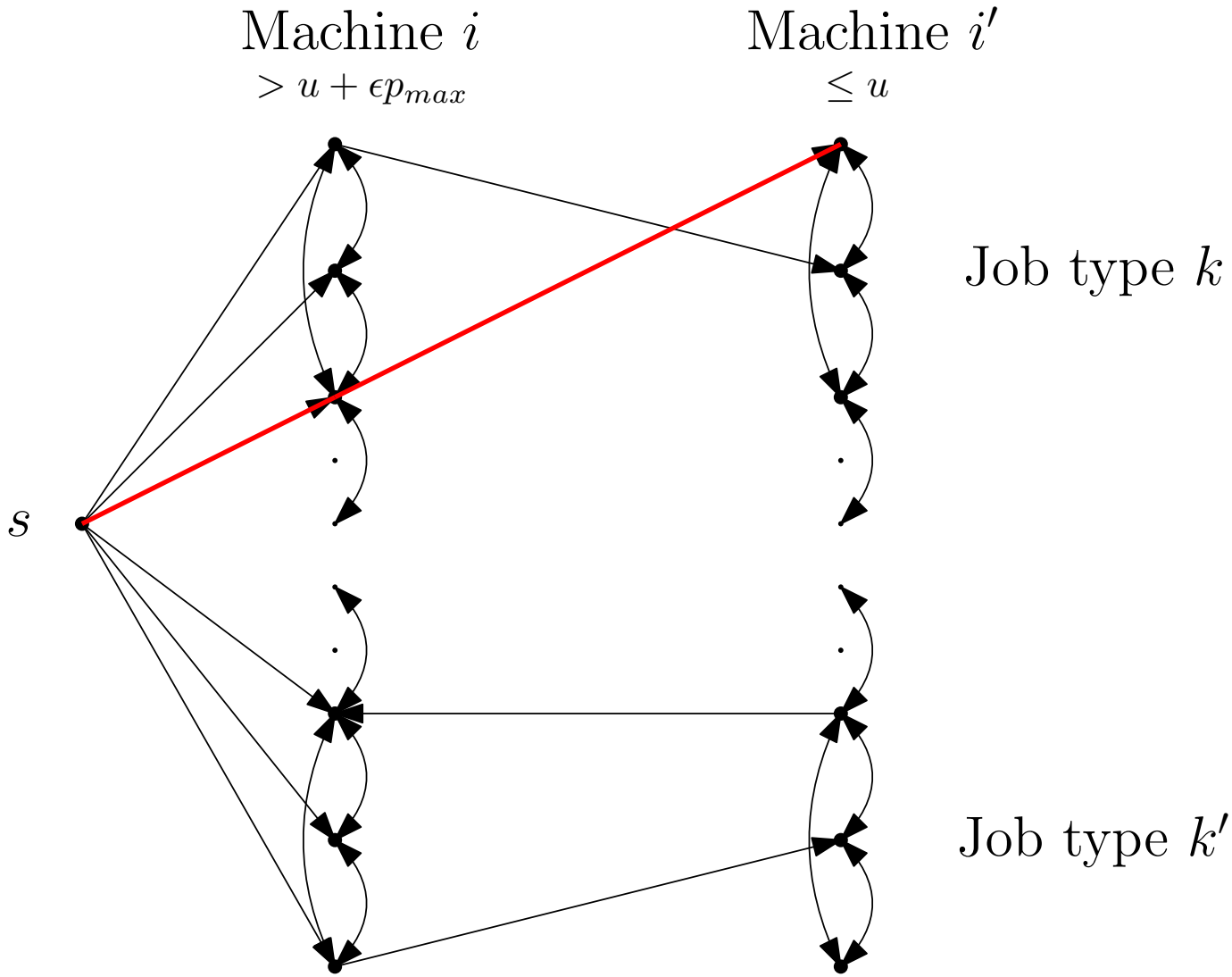
- Graph on $n + 1$ nodes:
  - **Source node** s
  - **Slot node** associated with a slot $y_{i,k}$ and the processing time of the job assigned to this slot
- **Three types of arcs**:
  - Slots on the same machine form a clique with arcs of weight 0
  - An arc from s to every slot on an overloaded machine of weight 1
  - An arc from slot v to slot w on a different machine if: (1) both slots are associated with the same job type and (2) the job assigned to v is larger than the job assigned to w

Breadth First Search Formulation

- Graph on $n + 1$ nodes:
  - **Source node** s
  - **Slot node** associated with a slot $y_{i,k}$ and the processing time of the job assigned to this slot
- **Three types of arcs**:
  - Slots on the same machine form a clique with arcs of weight 0
  - An arc from s to every slot on an overloaded machine of weight 1
  - An arc from slot v to slot w on a different machine if: (1) both slots are associated with the same job type and (2) the job assigned to v is larger than the job assigned to w

Iteration: Find a path from s to a slot on a machine with load at most u and swap the jobs associated to the final slot and the slot before that

Breadth First Search - Visualization



Rounding the slot-MILP

Lemma

Given an optimal solution (x, y) to the slot-MILP, in time $n^{O(1)}$ we can compute an integral solution (x', y) to slot-MILP such that for each machine $i \in \mathcal{M}$ we have:

$$\sum_{j \in \mathcal{J}} p_j x'_{i,j} \leq u + \epsilon \cdot p_{\max}.$$

Rounding the slot-MILP

Lemma

Given an optimal solution (x, y) to the slot-MILP, in time $n^{O(1)}$ we can compute an integral solution (x', y) to slot-MILP such that for each machine $i \in \mathcal{M}$ we have:

$$\sum_{j \in \mathcal{J}} p_j x'_{i,j} \leq u + \epsilon \cdot p_{\max}.$$

Proof.

1. **Existence of swappable jobs:** In each iteration a pair of swappable jobs can be found in time $O(n^2)$.

Rounding the slot-MILP

Lemma

Given an optimal solution (x, y) to the slot-MILP, in time $n^{O(1)}$ we can compute an integral solution (x', y) to slot-MILP such that for each machine $i \in \mathcal{M}$ we have:

$$\sum_{j \in \mathcal{J}} p_j x'_{i,j} \leq u + \epsilon \cdot p_{\max}.$$

Proof.

1. **Existence of swappable jobs:** In each iteration a pair of swappable jobs can be found in time $O(n^2)$.
2. **Number of swaps:** After at most $O(n^3)$ iterations every machine satisfies the load upper bound.

□

Putting together the pieces

- Solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Putting together the pieces

- Solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.
- Round the solution to the slot-MILP in running time $n^{O(1)}$

Putting together the pieces

- Solve the slot-MILP in time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.
- Round the solution to the slot-MILP in running time $n^{O(1)}$

Theorem (Additive polynomial time approximation scheme)

For any $\epsilon > 0$, we can find a solution for the makespan minimization, Santa Claus or envy-minimizing Santa Claus problem with a value ALG satisfying

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

in running time $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$.

Summary

- Consider **additive approximation schemes** for load balancing problems

Summary

- Consider **additive approximation schemes** for load balancing problems
- Additive approximation schemes in **two steps**: (1) solve the slot-MILP and (2) round the solution using local search

Summary

- Consider **additive approximation schemes** for load balancing problems
- Additive approximation schemes in **two steps**: (1) solve the slot-MILP and (2) round the solution using local search
- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem

Summary

- Consider **additive approximation schemes** for load balancing problems
- Additive approximation schemes in **two steps**: (1) solve the slot-MILP and (2) round the solution using local search
- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem
- **Improvement:** Using an alternative structural property we can improve the running time

Extensions and Improvements

- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem:

Extensions and Improvements

- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem:
  - Machine load lower bound ℓ needs to be considered

Extensions and Improvements

- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem:
  - Machine load lower bound ℓ needs to be considered
  - When rounding the solution we need to fix underloaded machines to ensure that each load satisfies:

$$\geq \ell - \epsilon p_{\max}$$

Extensions and Improvements

- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem:
  - Machine load lower bound ℓ needs to be considered
  - When rounding the solution we need to fix underloaded machines to ensure that each load satisfies:

$$\geq \ell - \epsilon p_{\max}$$

- **Improvement:** Using an alternative structural property we can improve the running time

Extensions and Improvements

- **Extensions:** Same idea gives an additive approximation scheme for Santa Claus and envy-minimizing Santa Claus problem:
  - Machine load lower bound ℓ needs to be considered
  - When rounding the solution we need to fix underloaded machines to ensure that each load satisfies:

$$\geq \ell - \epsilon p_{\max}$$

- **Improvement:** Using an alternative structural property we can improve the running time

Theorem

For any $\epsilon > 0$, we can find a solution for the makespan minimization, Santa Claus or envy-minimizing Santa Claus problem with a value ALG satisfying

$$|\mathrm{ALG} - \mathrm{OPT}| \leq \epsilon p_{\max}$$

in running time $m^2\left(\frac{n}{\epsilon^2}\right)^{O(1/\epsilon)}$.

Future Research

- **Improvement** of running time to obtain additive efficient polynomial time approximation schemes

Future Research

- **Improvement** of running time to obtain additive efficient polynomial time approximation schemes
- **Extension** of additive approximation schemes to other problems, e.g. sum of machine tardiness

Thank you

References I


N. Alon et al. “Approximation schemes for scheduling on parallel machines”. In: *Journal of Scheduling* 1 (1998), pp. 55–66.


Lin Chen, Klaus Jansen, and Guochuan Zhang. “On the optimality of exact and approximation algorithms for scheduling problems”. In: *J. Comput. Syst. Sci.* 96 (2018), pp. 1–32. DOI: 10.1016/j.jcss.2018.03.005. URL: https://doi.org/10.1016/j.jcss.2018.03.005.


M.R. Garey and D.S. Johnson. ““Strong” NP-completeness results: motivation, examples, and implications”. In: *Journal of the ACM* 25 (1978), pp. 499–508.


D.S. Hochbaum and D.B. Shmoys. “Using dual approximation algorithms for scheduling problems: Theoretical and practical results”. In: *Journal of the ACM* 34 (1987), pp. 144–162.

References II


Dorit S Hochbaum. "Various notions of approximations: Good, better, best and more". In: *Approximation algorithms for NP-hard problems* (1997).


K. Jansen. "An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables". In: *SIAM Journal on Discrete Mathematics* 24 (2010), pp. 457–485.


K. Jansen, K-M. Klein, and J. Verschae. "Closing the Gap for Makespan Scheduling via Sparsification Techniques". In: *Mathematics of Operations Research* (2020), to be published.