An (almost) optimal approximation scheme for minimum makespan scheduling

Klaus Jansen [1] Kim-Manuel Klein [1] **José Verschae**[2]

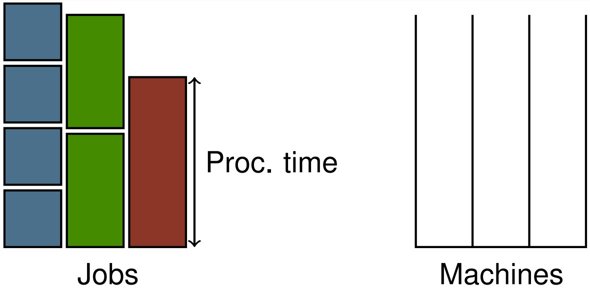
[1]University of Kiel
[2]Pontificia Universidad Católica de Chile

ADGO 2016

Problem Definition

Minimum Makespan Scheduling

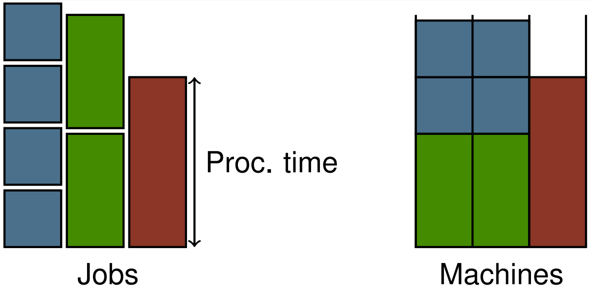
- n jobs.
- p_j: processing time job j.
- m machines.



Problem Definition

Minimum Makespan Scheduling

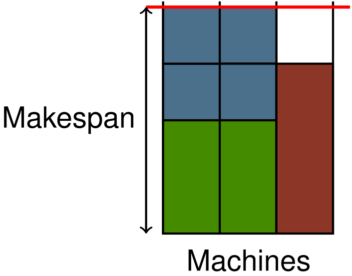
- n jobs.
- p_j: processing time job j.
- m machines.



Problem Definition

Minimum Makespan Scheduling

- n jobs.
- p_j: processing time job j.
- m machines.
- Objective: Minimize makespan (maximum machine load).



What do we seek?

Theorem

Determine the optimum solution is (strongly) NP-hard.

$\rightarrow$ Seeking poly-time (optimal) algorithms is unrealistic...

What do we seek?

Theorem

Determine the optimum solution is (strongly) NP-hard.

$\rightarrow$ Seeking poly-time (optimal) algorithms is unrealistic...

Definition

An algorithm is α-approximate if for each instance I then

$$\mathit{cost}(\mathrm{ALG}_I) \leq \underbrace{\alpha}_{\text{approx. factor}} \mathrm{OPT}_I.$$

What do we seek?

Theorem

Determine the optimum solution is (strongly) NP-hard.

$\rightarrow$ Seeking poly-time (optimal) algorithms is unrealistic...

Definition

An algorithm is α-approximate if for each instance I then

$$\mathit{cost}(\mathrm{ALG}_I) \leq \underbrace{\alpha}_{\text{approx. factor}} \mathrm{OPT}_I.$$

Main Questions

- *Classic question*: What is the lowest approx factor achievable in poly-time?
- *Modern question*: Given an approximation factor, what is the best possible running time?

Classic Question

Definition

A family of algorithms $(\mathcal{A}_\varepsilon)_{\varepsilon>0}$ is a *Polynomial Time Approximation Scheme (PTAS)* if, for all $\varepsilon > 0$,

- $\mathcal{A}_\varepsilon$ is a poly-time algorithm, and
- $\mathcal{A}_\varepsilon$ is $(1+\varepsilon)$-approximate.

Classic Question

Definition

A family of algorithms $(\mathcal{A}_\varepsilon)_{\varepsilon>0}$ is a *Polynomial Time Approximation Scheme (PTAS)* if, for all $\varepsilon > 0$,

- $\mathcal{A}_\varepsilon$ is a poly-time algorithm, and
- $\mathcal{A}_\varepsilon$ is $(1+\varepsilon)$-approximate.

Previous Literature

- A greedy algorithm is 4/3-approximate.

  [Graham '66 + '69]

- There is a PTAS with running time $n^{\widetilde{O}(\frac{1}{\varepsilon^2})}$.

  [Hochbaum & Shmoys '87]

Classic Question

Definition

A family of algorithms $(\mathcal{A}_\varepsilon)_{\varepsilon>0}$ is a *Polynomial Time Approximation Scheme (PTAS)* if, for all $\varepsilon > 0$,

- $\mathcal{A}_\varepsilon$ is a poly-time algorithm, and
- $\mathcal{A}_\varepsilon$ is $(1+\varepsilon)$-approximate.

Previous Literature

- A greedy algorithm is 4/3-approximate.

  [Graham '66 + '69]
- There is a PTAS with running time $n^{\widetilde{O}(\frac{1}{\varepsilon^2})}$.

  [Hochbaum & Shmoys '87]

Classic Question: Solved!

Modern Question

Reinterpretation: What is the best running time possible for a PTAS?

Known Algorithms

There is a PTAS with running time (roughly):

- $n^{\widetilde{O}(\frac{1}{\varepsilon^2})}$ [Hochbaum & Shmoys '87]
- $n^{\widetilde{O}(\frac{1}{\varepsilon})}$ [Leung 97]

- $2^{(\frac{1}{\varepsilon})^{\widetilde{O}(\frac{1}{\varepsilon})}} + n$ [Alon et al. '98 & H. & S. '96]
- $2^{\widetilde{O}(\frac{1}{\varepsilon^2})} + n \log n$ [Jansen '10]

Modern Question

Reinterpretation: What is the best running time possible for a PTAS?

Known Algorithms

There is a PTAS with running time (roughly):

- $n^{\widetilde{O}(\frac{1}{\varepsilon^2})}$ [Hochbaum & Shmoys '87]
- $n^{\widetilde{O}(\frac{1}{\varepsilon})}$ [Leung 97]

- $2^{(\frac{1}{\varepsilon})^{\widetilde{O}(\frac{1}{\varepsilon})}} + n$ [Alon et al. '98 & H. & S. '96]
- $2^{\widetilde{O}(\frac{1}{\varepsilon^2})} + n \log n$ [Jansen '10]

Lower Bounds

- If $P \neq NP$, no PTAS can have a polynomial dependency on $\frac{1}{\varepsilon}$ [Folkclore]
- If the *Exponential Time Hypothesis* holds, there is no PTAS with running time $2^{(\frac{1}{\varepsilon})^{1-\delta}} + n^{O(1)}$. [Chen et al. '13]

Our Main Result

Theorem

Minimum makespan scheduling admits a PTAS with running time

$$2^{\widetilde{O}(\frac{1}{\varepsilon})} + n^{O(1)}.$$

Our Main Result

Theorem

Minimum makespan scheduling admits a PTAS with running time

$$2^{\widetilde{O}(\frac{1}{\varepsilon})} + n^{O(1)}.$$

Lower Bound

- If the *Exponential Time Hypothesis* holds, there is no PTAS with running time $2^{(\frac{1}{\varepsilon})^{1-\delta}} + n^{O(1)}$. [Chen et al. '13]

General Strategy

General Scheme for designing a PTAS:

1. Round instance $\rightsquigarrow$ $(1 + \varepsilon)$ multiplicative loss in objective.
2. Show that optimal solution of rounded instances has a "nice" structure.
3. Look for optimal solution with such structure.

General Strategy

General Scheme for designing a PTAS:

1. Round instance $\rightsquigarrow$ $(1 + \varepsilon)$ multiplicative loss in objective.
2. Show that optimal solution of rounded instances has a “nice” structure.
3. Look for optimal solution with such structure.

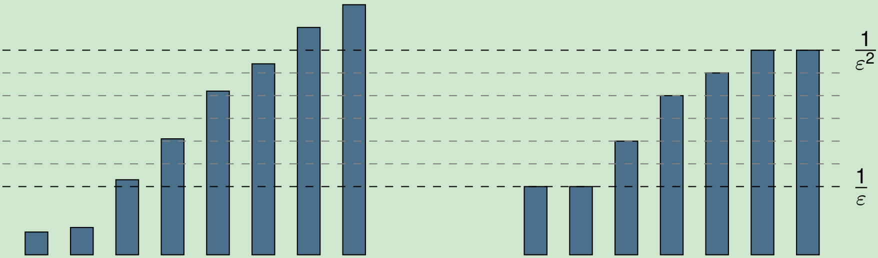
Rounding

Lemma

After rounding (and scaling) OPT $\in \{1, \ldots, (2/\varepsilon)^2\}$ *and the sizes of jobs belong to a set P such that:*

- $P \subseteq \{\frac{1}{\varepsilon}, \frac{1}{\varepsilon} + 1, \ldots, \frac{1}{\varepsilon^2}\}$ *and,*
- $|P| \leq \widetilde{O}(\frac{1}{\varepsilon})$.

Example



Let T be a guessed value for OPT.

Configurations

- Let T be a guessed value of OPT.
- Each machine gets at most $2/\varepsilon$ jobs.

Configurations

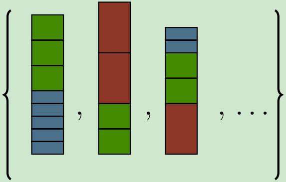
- Let T be a guessed value of OPT.
- Each machine gets at most $2/\varepsilon$ jobs.

Configurations

A *configuration* is a one-machine schedule with total size $\leq T$.
Obs: If K is the set of all configurations, then

$$|K| \leq \left(\frac{2}{\varepsilon}\right)^{|P|} = 2^{\widetilde{O}(\frac{1}{\varepsilon})}.$$

Example

$K =$ 

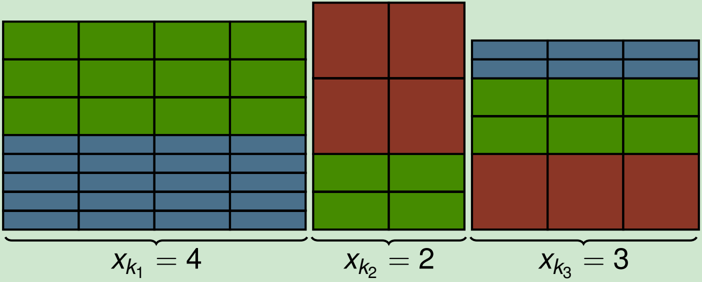
Compact description of a schedule

Multiple machines

We consider vector $(x_k)_{k \in K} \in \{0, \ldots, m\}^{|K|}$, where

$$x_k : \text{number of machines following configuration } k \in K.$$

Example



Integer Programming Formulation

Observation

The vector $(x_k)_{k \in K}$ belongs to the system

$$\begin{aligned} \sum_{k \in K} x_k &= m \\ \sum_{k \in K} k_p x_k &= n_p \quad \text{for all } p \in P \\ x &\in \mathbb{Z}_{\geq 0}^K \end{aligned}$$

Integer Programming Formulation

Observation

The vector $(x_k)_{k \in K}$ belongs to the system

$$\left.\begin{aligned} \sum_{k \in K} x_k &= m \\ \sum_{k \in K} k_p x_k &= n_p \quad \text{for all } p \in P \\ x &\in \mathbb{Z}_{\geq 0}^{K} \end{aligned}\right\} \quad \boxed{\begin{array}{l} \text{\# of constraints: } \widetilde{O}(\frac{1}{\varepsilon}) \\ \text{\# variables: } 2^{\widetilde{O}(\frac{1}{\varepsilon})} \end{array}}$$

Solving the IP: First Approach

Direct method [Alon et al. '98]

Use the following result:

Theorem (Kannan '87)

An integer program with d variables can be solved in time $2^{\widetilde{O}(d)}s$ (where s is the length of the input).

In our case $d = |K| = 2^{\widetilde{O}(\frac{1}{\varepsilon})}$ and thus the running time is at least

$$d^d = 2^{2^{\widetilde{O}(\frac{1}{\varepsilon})}}$$

Solving the IP: First Approach

Direct method [Alon et al. '98]

Use the following result:

Theorem (Kannan '87)

An integer program with d variables can be solved in time $2^{\widetilde{O}(d)} s$ (where s is the length of the input).

In our case $d = |K| = 2^{\widetilde{O}(\frac{1}{\varepsilon})}$ and thus the running time is at least

$$d^d = 2^{2^{\widetilde{O}(\frac{1}{\varepsilon})}} \leftarrow \text{doubly exponential!}$$

Solving the IP: First Approach

Direct method [Alon et al. '98]

Use the following result:

Theorem (Kannan '87)

An integer program with d variables can be solved in time $2^{\widetilde{O}(d)} s$ (where s is the length of the input).

In our case $d = |K| = 2^{\widetilde{O}(\frac{1}{\varepsilon})}$ and thus the running time is at least

$$d^d = 2^{2^{\widetilde{O}(\frac{1}{\varepsilon})}} \leftarrow \text{doubly exponential!}$$

If we only could decrease the number of variables...

Solving the IP: Second Approach

Guess the support [Jansen '10]

Theorem (Eisenbrand & Shmonin 2006)

A problem $\{c^t x : Ax = b, x \in \mathbb{Z}_{\geq 0}\}$
where A has h rows, admits an optimal solution x^* *with*

$$|supp(x^*)| \leq O(h \log(h + s)).$$

For our case:

- $h = |P| + 1 \approx \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon})$, and
- $\mathrm{supp}(x^*) \leq \widetilde{O}(\frac{1}{\varepsilon})$

Solving the IP: Second Approach

Guess the support [Jansen '10]

Theorem (Eisenbrand & Shmonin 2006)

A problem $\{c^t x : Ax = b, x \in \mathbb{Z}_{\geq 0}\}$
where A has h rows, admits an optimal solution x^* *with*

$$|supp(x^*)| \leq O(h \log(h+s)).$$

For our case:

- $h = |P| + 1 \approx \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon})$, and
- $\text{supp}(x^*) \leq \widetilde{O}(\frac{1}{\varepsilon})$

Idea:

1. Try each possible support S: there are $\binom{|K|}{\widetilde{O}(\frac{1}{\varepsilon})} = 2^{\widetilde{O}(\frac{1}{\varepsilon^2})}$ many possibilities.
2. For each possibility solve the IP restricted to those variables with Kannan's algorithm.
3. Total running time: $2^{\widetilde{O}(\frac{1}{\varepsilon^2})}$.

Solving the IP: Third Approach

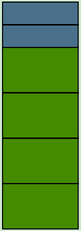
Understanding the Optimum

Definition

A configuration k is *complex* if contains more than $\log(1/\varepsilon^2)$ different sizes; o.w. is *simple*.

Example

$\leq \log(1/\varepsilon^2)$ sizes (colors)



Simple

$> \log(1/\varepsilon^2)$ sizes (colors)

Complex

$\boxed{\log(\frac{1}{\varepsilon^2}) = 3}$

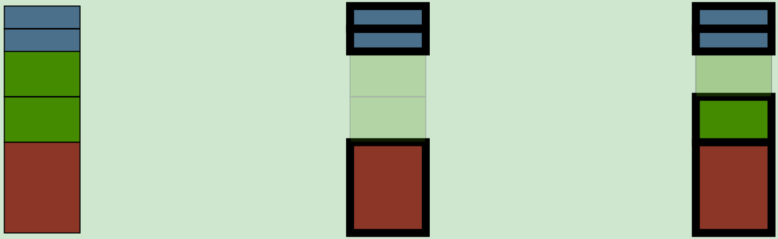
Solving the IP: Third Approach

Understanding the Optimum

Definition (Informal)

A "subconfiguration" of a configuration k is called *maximal* if it contains all possible jobs of each taken size.

Example



Original Configuration

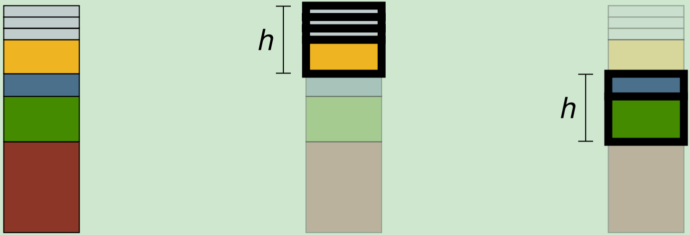
Maximal Subconfiguration

Non-Maximal Subconfiguration

Lemma

Every complex conf. $k \in K$ contains two maximal subconfigurations k_1, k_2 s.t. the total size of k_1 and k_2 coincide.

Example



Complex Configuration k

Subconfiguration k_1

Subconfiguration k_2

Lemma

Every complex conf. $k \in K$ contains two maximal subconfigurations k_1, k_2 s.t. the total "size" of k_1 and k_2 coincide.

Proof.

- Let $C > \log \frac{1}{\varepsilon^2}$ be the number of sizes (colors) in k.
- Number of maximal subconfigurations $2^C \geq \frac{1}{\varepsilon^2}$.
- The total size of each configuration belongs to $\{1, 2, \ldots, \frac{1}{\varepsilon^2}\}$.
- $\Rightarrow$ there must be two maximal subconfigurations of same total size.

$\square$

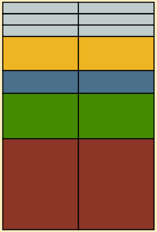
Solving the IP: Third Approach

Understanding the Optimum

Lemma (Sparsification Lemma (informal))

If a complex configuration is taken twice in a solution, then we can replace it by two other "less complex" configurations.

Proof.



□

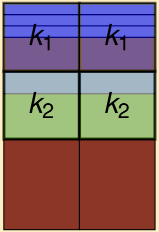
Solving the IP: Third Approach

Understanding the Optimum

Lemma (Sparsification Lemma (informal))

If a complex configuration is taken twice in a solution, then we can replace it by two other "less complex" configurations.

Proof.



□

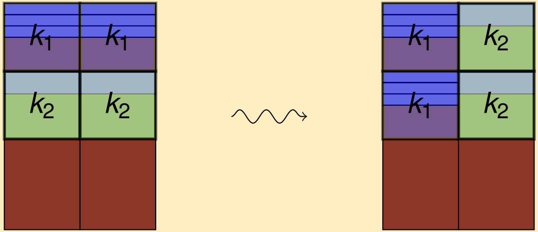
Solving the IP: Third Approach

Understanding the Optimum

Lemma (Sparsification Lemma (informal))

If a complex configuration is taken twice in a solution, then we can replace it by two other "less complex" configurations.

Proof.



□

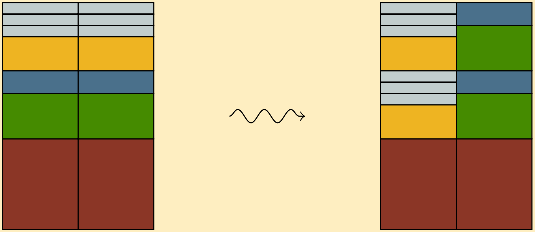
Solving the IP: Third Approach

Understanding the Optimum

Lemma (Sparsification Lemma (informal))

If a complex configuration is taken twice in a solution, then we can replace it by two other "less complex" configurations.

Proof.



□

Solving the IP: Third Approach

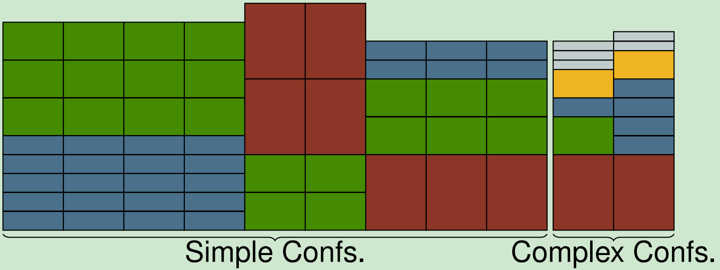
Understanding the Optimum

Theorem (Thin Solutions)

If the IP is not empty, then there is a solution x^* such that:

- *At most $\widetilde{O}(\frac{1}{\varepsilon})$ machines get complex configurations*
- *Each complex configuration is used at most once*
- $|supp(x^*)| \leq \widetilde{O}(\frac{1}{\varepsilon})$.

Example



Solving the IP: Third Approach

Part 1:

1. Guess jobs assigned to complex configurations, and number of complex machines.
2. Solve that subinstance optimally (with a Dynamic Program).

Solving the IP: Third Approach

Part 1:

1. Guess jobs assigned to complex configurations, and number of complex machines.
2. Solve that subinstance optimally (with a Dynamic Program).

Part 2: Remaining instance.

1. Guess the (simple!) configurations that x uses: there are $\binom{2^{\log^2(\frac{1}{\varepsilon})}}{\widetilde{O}(\frac{1}{\varepsilon})} = 2^{\widetilde{O}(\frac{1}{\varepsilon})}$ many possibilities.
2. For each possibility solve the IP restricted to those variables with Kannan's algorithm.

Part 1:

1. Guess jobs assigned to complex configurations, and number of complex machines.
2. Solve that subinstance optimally (with a Dynamic Program).

Part 2: Remaining instance.

1. Guess the (simple!) configurations that x uses: there are $\binom{2^{\log^2(\frac{1}{\varepsilon})}}{\widetilde{O}(\frac{1}{\varepsilon})} = 2^{\widetilde{O}(\frac{1}{\varepsilon})}$ many possibilities.
2. For each possibility solve the IP restricted to those variables with Kannan's algorithm.

Total running time: $2^{\widetilde{O}(\frac{1}{\varepsilon})}$

Summary of Results

1. The minimum makespan problem can be solved in time $2^{\widetilde{O}(\frac{1}{\varepsilon})} + \text{poly}(n)$.
2. The result is best possible up to logarithmic factors in the exponent (assuming ETH).
3. Possibility to apply the same idea to other problems: in particular for the related machines makespan scheduling.