Oracle-based approximation algorithms for the discrete resource sharing scheduling problem

Marin Bougeret, Pierre-François Dutot, Denis Trystram

Laboratoire LIG

13 May, University of Tennessee

1. *PTAS* design techniques
   - Some classical techniques
   - The oracle formalism

2. The DRSSP problem

3. Oracle approximation
   - Guessing the correct oracle answer
   - Second guess : convenient subset
   - Guess approximation

1. *PTAS* design techniques
   - Some classical techniques
   - The oracle formalism

2. The DRSSP problem

3. Oracle approximation
   - Guessing the correct oracle answer
   - Second guess : convenient subset
   - Guess approximation

The main techniques...

Some of the main *PTAS* design techniques [SW00]:

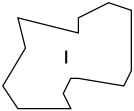
- structuring the input
- structuring the output ("extending partial small size solutions")
- structuring the execution of an algorithm ("trimmed algorithm")

Structuring the input

Given in instance I, the main ("polynomial") steps are:

- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I

Figure from [SW00]

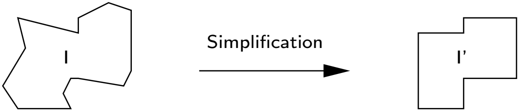


Structuring the input

Given in instance I, the main ("polynomial") steps are:

- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I

Figure from [**SW00**]

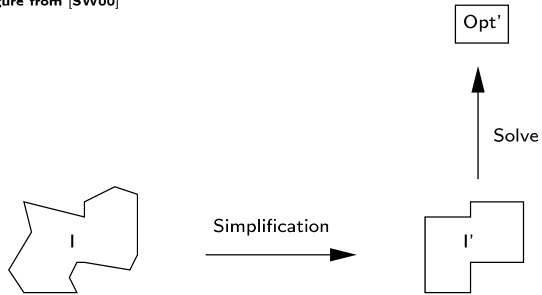


Structuring the input

Given in instance I, the main ("polynomial") steps are:

- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I

Figure from [**SW00**]

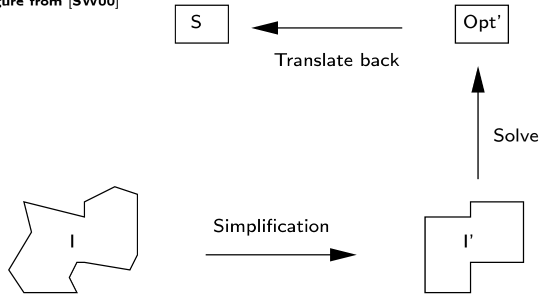


Structuring the input

Given in instance I, the main ("polynomial") steps are:

- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I

Figure from [SW00]

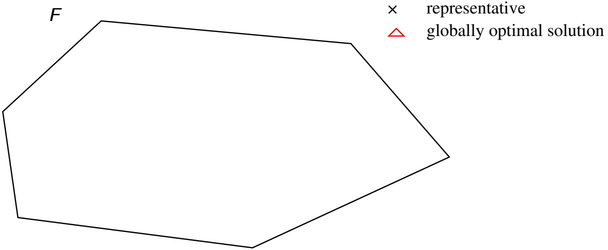


Structuring the output

Given in instance I, the main ("polynomial") steps are:

- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take the best**: select the best of all representatives as the final solution S

figure from [SW00]

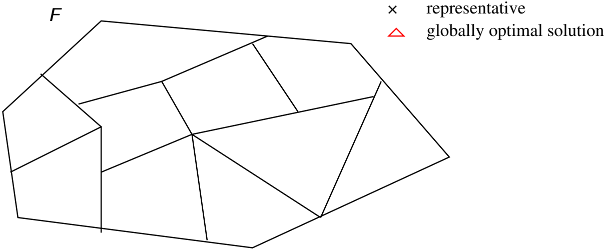


Structuring the output

Given in instance I, the main ("polynomial") steps are:

- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take the best**: select the best of all representatives as the final solution S

figure from [SW00]

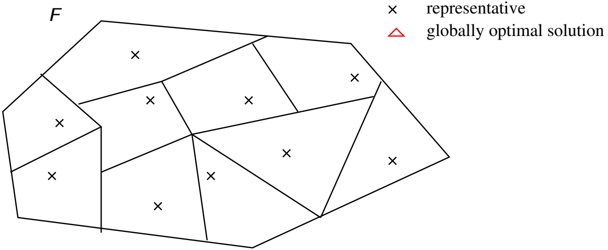


Structuring the output

Given in instance I, the main ("polynomial") steps are:

- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take the best**: select the best of all representatives as the final solution S

figure from [SW00]

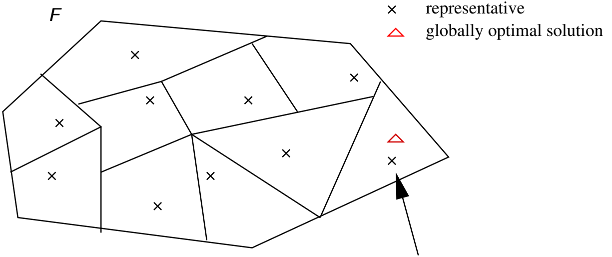


Structuring the output

Given in instance I, the main ("polynomial") steps are:

- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take the best**: select the best of all representatives as the final solution S

figure from [SW00]



Structuring the execution of an algorithm

Given in instance I, perform a polynomial number of "meta" steps. At each step:

- **extend**: extend every partial solution of the current set
- **collapse**: according to a previously defined "grid"', collapse all the partial solutions which are in the same "box"

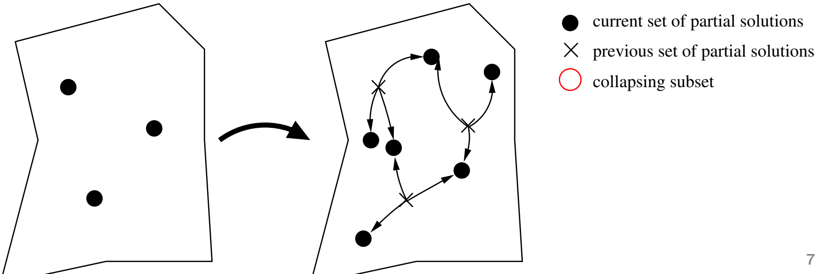
take the best: After the last step, we get solutions for the original problem. Select the best of all these solutions.

Structuring the execution of an algorithm

Given in instance I, perform a polynomial number of "meta" steps.
At each step:

- **extend**: extend every partial solution of the current set
- **collapse**: according to a previously defined "grid"', collapse all the partial solutions which are in the same "box"

take the best: After the last step, we get solutions for the original problem. Select the best of all these solutions.

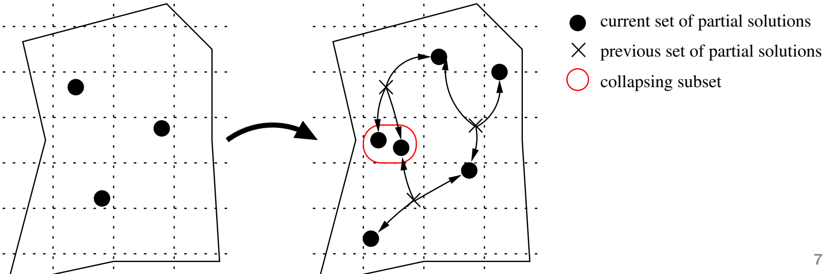


Structuring the execution of an algorithm

Given in instance I, perform a polynomial number of "meta" steps. At each step:

- **extend**: extend every partial solution of the current set
- **collapse**: according to a previously defined "grid"', collapse all the partial solutions which are in the same "box"

take the best: After the last step, we get solutions for the original problem. Select the best of all these solutions.

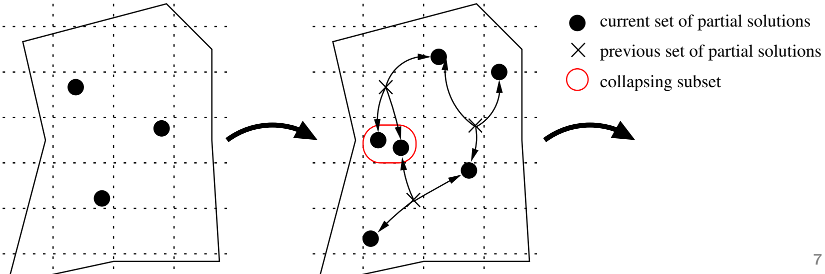


Structuring the execution of an algorithm

Given in instance I, perform a polynomial number of "meta" steps. At each step:

- **extend**: extend every partial solution of the current set
- **collapse**: according to a previously defined "grid"', collapse all the partial solutions which are in the same "box"

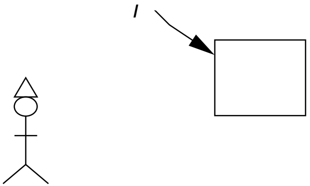
take the best: After the last step, we get solutions for the original problem. Select the best of all these solutions.



Oracle based approach

This formalism is directly inspired from complexity theory. Given in instance I, the main ("polynomial") steps are:

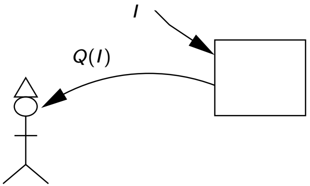
- **choose the question**: choose an "interesting" property P
- ask a question $Q(I)$ to the (reliable) oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- **find a solution using the answer**: A provides $S(r^*) \leq \rho Opt$
- **without the oracle**: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

This formalism is directly inspired from complexity theory. Given in instance I, the main ("polynomial") steps are:

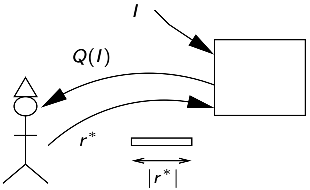
- **choose the question**: choose an "interesting" property P
- ask a question $Q(I)$ to the (reliable) oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- **find a solution using the answer**: A provides $S(r^*) \leq \rho Opt$
- **without the oracle**: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

This formalism is directly inspired from complexity theory. Given in instance I, the main ("polynomial") steps are:

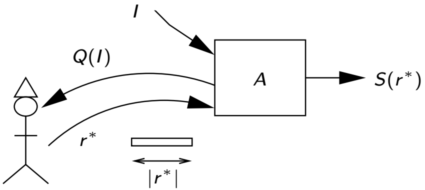
- **choose the question**: choose an "interesting" property P
- ask a question $Q(I)$ to the (reliable) oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- **find a solution using the answer**: A provides $S(r^*) \leq \rho Opt$
- **without the oracle**: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

This formalism is directly inspired from complexity theory. Given in instance I, the main ("polynomial") steps are:

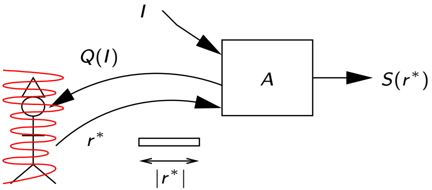
- **choose the question**: choose an "interesting" property P
- ask a question $Q(I)$ to the (reliable) oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- **find a solution using the answer**: A provides $S(r^*) \leq \rho Opt$
- **without the oracle**: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

This formalism is directly inspired from complexity theory. Given in instance I, the main ("polynomial") steps are:

- **choose the question**: choose an "interesting" property P
- ask a question $Q(I)$ to the (reliable) oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- **find a solution using the answer**: A provides $S(r^*) \leq \rho Opt$
- **without the oracle**: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

Thus, the obtained algorithm (without oracle):

- is a ρ approximation
- has a computational complexity in $O(t_A * 2^{|r^*|})$

Generally, we can choose the size $|r^*|$ (leading to different ρ), leading to approximation schemes.

The answer size is crucial !

Beside efficient (compact) representation, we will look into lossy compression.

Oracle based approach

Thus, the obtained algorithm (without oracle):

- is a ρ approximation
- has a computational complexity in $O(t_A * 2^{|r^*|})$

Generally, we can choose the size $|r^*|$ (leading to different ρ), leading to approximation schemes.

The answer size is crucial !

Beside efficient (compact) representation, we will look into lossy compression.

Oracle approach Vs structuring the output

When asking a particular type of "questions", the oracle formalism can be equivalent to the output structuring technique. An example for $P||C_{max}$:

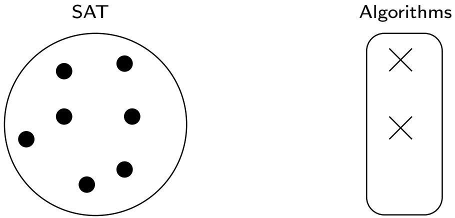
- question : where do you schedule the biggest task (in an optimal solution)?
- provide a solution for all the possible oracle answer $r \iff$ provide a solution for every district
- the oracle answer r^* indicates a district containing an optimal solution

However, these are very special cases.

1. *PTAS* design techniques
   - Some classical techniques
   - The oracle formalism
2. The DRSSP problem
3. Oracle approximation
   - Guessing the correct oracle answer
   - Second guess : convenient subset
   - Guess approximation

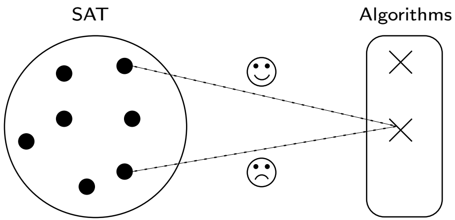
Introduction

- finite benchmark of instances: allows comparisons between algorithms
- set of algorithms
- goal: minimize the time needed to solve all the instances from the benchmark
- more than selection: combination of algorithms



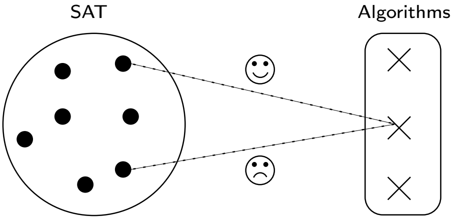
Introduction

- finite benchmark of instances: allows comparisons between algorithms
- set of algorithms
- goal: minimize the time needed to solve all the instances from the benchmark
- more than selection: combination of algorithms



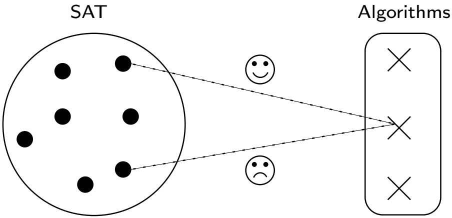
Introduction

- finite benchmark of instances: allows comparisons between algorithms
- set of algorithms
- goal: minimize the time needed to solve all the instances from the benchmark
- more than selection: combination of algorithms



Introduction

- finite benchmark of instances: allows comparisons between algorithms
- set of algorithms
- goal: minimize the time needed to solve all the instances from the benchmark
- more than selection: combination of algorithms



Introduction

What me mean by combination:

- one instance may be treated by several algorithms in parallel
- when a solution of an instance is found, everyone is aware
- but, the solution for an instance cannot be merged from partial solutions provided by different algorithms

Algorithm are parallel.
Parallel task model : moldable.

Introduction

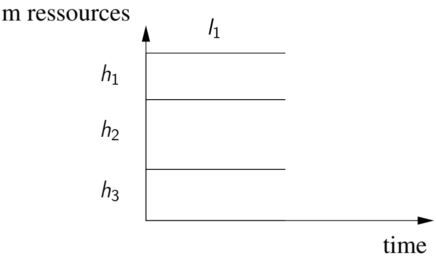
Outline :

- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$

Introduction

Outline :

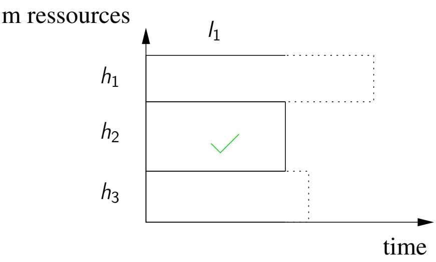
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$



Introduction

Outline :

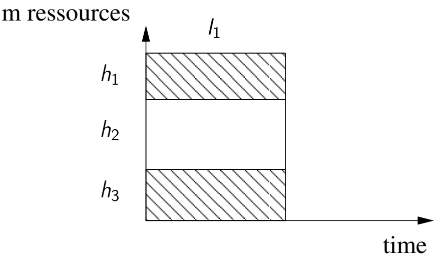
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$



Introduction

Outline :

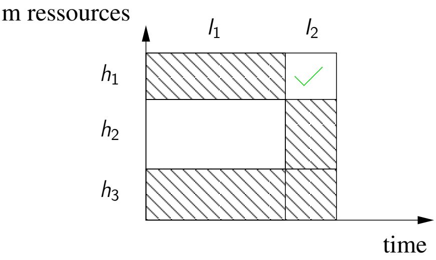
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$



Introduction

Outline :

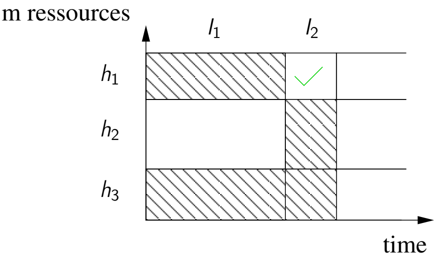
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$



Introduction

Outline :

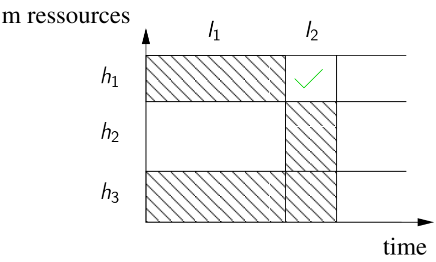
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$

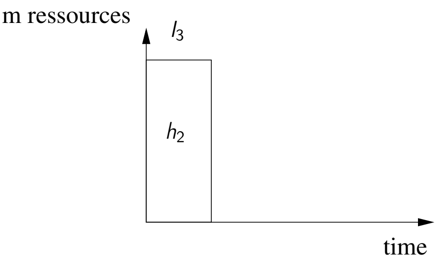


Introduction

Outline :

- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$

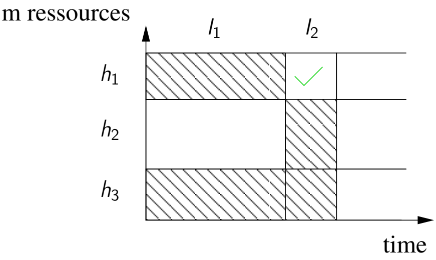


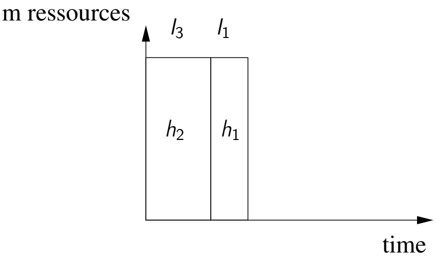


Introduction

Outline :

- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$

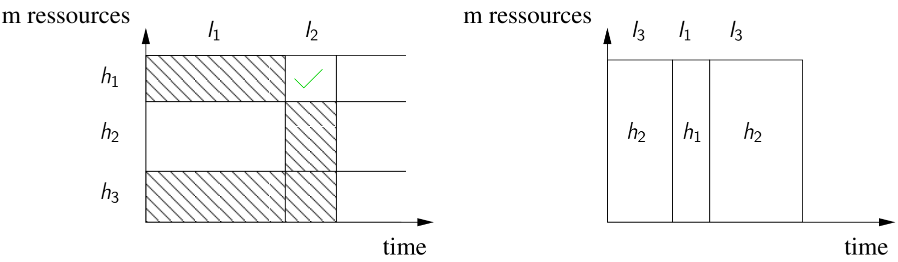




Introduction

Outline :

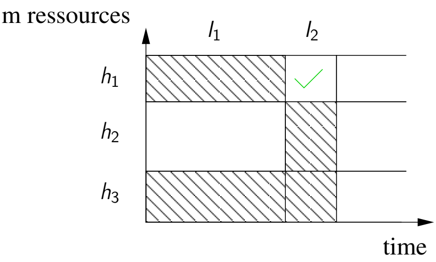
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$

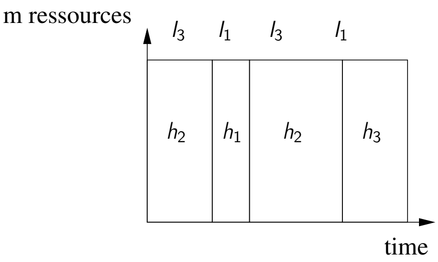


Introduction

Outline :

- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$

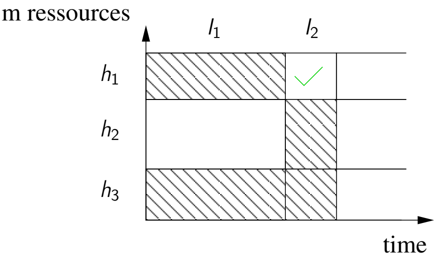


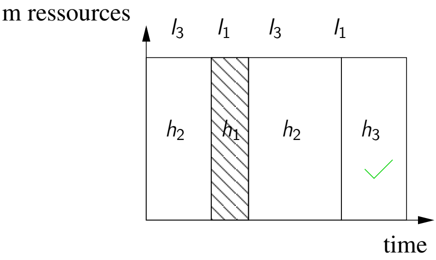


Introduction

Outline :

- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for instance I_j, algorithm h_i and p resources, the time cost is $C(h_i, I_j, p)$





Introduction

Context :

- hybridation, algorithm portfolios
- two of the existing techniques : time sharing Vs space sharing

Space sharing assumptions (for a fixed problem P):

- a portfolio of algorithm for P is given
- there exists a finite set I of *representative* input of P
- the time needed by every algorithm to solve every instance of I is known a priori !
- the goal is to minimize the mean execution time for an instance of I

Definition of the dRSSP

Input of the discrete Resource Sharing Scheduling Problem:

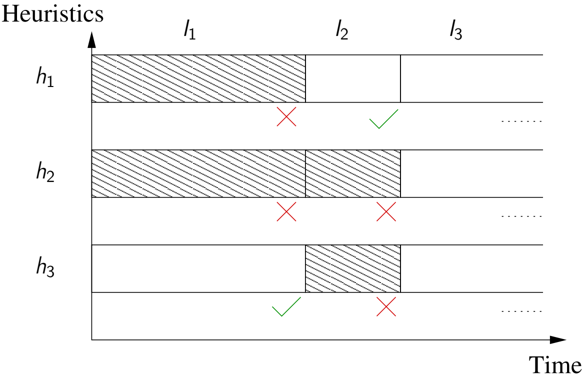
- a finite set of instances $I = \{I_1, \ldots, I_n\}$
- a finite set of heuristics $H = \{h_1, \ldots, h_k\}$
- m identical resources
- a cost $C(h_i, I_j, p) \in R^+$ for each $I_j \in I$, $h_i \in H$ and $p \in \{1, \ldots, m\}$

Continuous version ($p \in R^+$) in [SFM06].

Definition of the dRSSP

Output : an allocation $S = (S_1, \ldots, S_k)$ such that:

- $S_i \in \{0, \ldots, m\}$
- $0 < \sum_{i=1}^{k} S_i \leq m$
- S minimizes $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{C(h_i, I_j, S_i) | S_i > 0\}$



A restricted version

We study a restricted version in which :

- the cost function is linear in the number of resources $C(h_i, I_j, S_i) = \frac{C(h_i, I_j)}{S_i}$
- each heuristic must use at least one processor ($S_i \geq 1$), (well chosen portfolio)

Remark : with only the first constraint, the problem is inaproximable within a constant factor (if $m < k$).

A simple greedy algorithm

We consider the mean-allocation (MA) algorithm which simply allocates $\lfloor \frac{m}{k} \rfloor$ resources to each heuristic.

Proposition

MA is a k approximation.

Notations (given a solution S):

- let $\sigma(j) = i_0 / \frac{C(h_{i_0}, I_j)}{S_{i_0}} = \min_{1 \leq i \leq k} \frac{C(h_i, I_j)}{S_i}$ be the index of the used heuristic for instance $j \in \{1, .., n\}$ in S
- let $T(I_j) = \frac{C(h_{\sigma(j)}, I_j)}{S_{\sigma(j)}}$ be the processing time of instance j in S

A simple greedy algorithm

Proof: Let $(a, b) \in \mathbb{N}^2$ such that $m = ak + b, b < k, a \geq 1$.
$\forall j \in \{1, .., n\}$:

$$\begin{aligned} T(I_j) \leq \frac{C(h_{\sigma*(j)}, I_j)}{S_{\sigma^*(j)}} &= \frac{S^*_{\sigma^*(j)}}{S_{\sigma^*(j)}} T^*(I_j) \\ &\leq \frac{m-(k-1)}{S_{\sigma^*(j)}} T^*(I_j) \\ &= \frac{ak+b-(k-1)}{a} T^*(I_j) \leq kT^*(I_j) \end{aligned}$$

Guess 1

As a first step, we ask the correct allotment for g heuristics.

Definition

Let $G_1 = (S_1^*, \ldots, S_g^*)$, for a fixed subset of g heuristics and a fixed optimal solution S^*.

Notice that $|G_1| = glog(m)$.

We need some notations :

- let $k' = k - g$ be the number of remaining heuristics
- let $s = \Sigma_{i=1}^{g} S_i^*$ the number of processors used in the guess
- let $m' = m - s$ the number of remaining processors
- let $(a', b') \in \mathbb{N}^2$ such that $m' = a'k' + b', b' < k'$

Guess 1

As a first step, we ask the correct allotment for g heuristics.

Definition

Let $G_1 = (S_1^*, \dots, S_g^*)$, for a fixed subset of g heuristics and a fixed optimal solution S^*.

Notice that $|G_1| = glog(m)$.
We need some notations :

- let $k' = k - g$ be the number of remaining heuristics
- let $s = \Sigma_{i=1}^{g} S_i^*$ the number of processors used in the guess
- let $m' = m - s$ the number of remaining processors
- let $(a', b') \in \mathbb{N}^2$ such that $m' = a'k' + b', b' < k'$

Algorithm MA^G

We consider the following MA^G algorithm (given any guess $G = (X_1, \ldots, X_g), X_i \geq 1$):

- allocate X_i processors to heuristic $h_i, i \in \{1, \ldots, g\}$
- applies MA on the k' others heuristics with the m' remaining processors

We will use this algorithm with $G = G_1$.

Analysis of MA^{G_1}

Proposition

MA^{G_1} is a $k - g$ approximation.

Proof:

- MA^{G_1} produces a valid solution because $a' \geq 1$
- for any instance j treated by a guessed heuristic in the optimal solution considered MA^{G_1} is even better than the optimal
- for the others, the analysis is the same as for the algorithm MA, and leads to the desired ratio

Algorithm MA_R^G

The ratio for instances treated by the guessed heuristics is unnecessarily good.
Thus, we consider mean-allocation-reassign (MA_R^G) algorithm (given any guess $G = (X_1, \ldots, X_g), X_i \geq 1$):

- allocates $X_i - \lfloor \frac{X_i}{\alpha} \rfloor$ processors to heuristic $h_i, i \in \{1, \ldots, g\}$
- applies MA on the k' others heuristics with the $m' + \Sigma_{i=1}^{g} \lfloor \frac{X_i}{\alpha} \rfloor$ remaining processors

Remark

- MA_R^G doesn't respect G
- maybe we asked the wrong question ?

Another analysis of MA

For any heuristic $h_i, i \in \{1, .., k\}$, let $T^*(h_i) = \Sigma_{j/\sigma^*(j)=i} T^*(I_j)$ be the "useful" computation time of heuristic i in the solution S^*.

$$
\begin{aligned}
T_{MA} &= \sum_{i=1}^{k} \sum_{j/\sigma^*(j)=i} T(I_j) \\
&\leq \sum_{i=1}^{k} \frac{S_i^*}{S_i} \sum_{j/\sigma^*(j)=i} T^*(I_j) \\
&= \sum_{i=1}^{k} \frac{S_i^*}{S_i} T^*(h_i) \\
&\leq Max_i(T^*(h_i)) \frac{m}{\lfloor \frac{m}{k} \rfloor} \\
&\leq Max_i(T^*(h_i))(2k-1)
\end{aligned}
$$

Guess 2

Definition

Let $G_2 = (S_1^*, \ldots, S_g^*)$, such that $T^*(h_1) \geq .. \geq T^*(h_g) \geq T^*(h_i), \forall i \in \{g+1, .., k\}$ in a fixed optimal solution S^*.

Notice that $|G_2| = glog(k) + glog(m)$.
We will use the algorithm MA^G with $G = G_2$.

Analysis of MA^{G_2}

Proposition

MA^{G_2} is a $\frac{k-1}{g}$ approximation.

Proof: We proceed as in the new analysis of *MA*:

$$
\begin{aligned}
T_{algo} &= \sum_{i=1}^{g} \sum_{j/\sigma^*(j)=i} T(I_j) + \sum_{i=g+1}^{k} \sum_{j/\sigma^*(j)=i} T(I_j) \\
&\leq \sum_{i=1}^{g} T^*(h_i) + \sum_{i=g+1}^{k} \frac{S_i^*}{S_i} T^*(h_i) \\
&= \sum_{i=1}^{k} T^*(h_i) + \sum_{i=g+1}^{k} (\frac{S_i^*}{S_i} - 1) T^*(h_i) \\
&= Opt + \underbrace{T^*(h_g)}_{\leq \frac{Opt}{g}} (\frac{m'}{a'} - k')
\end{aligned}
$$

Introduction

Goal: we want $\bar{G}$ smaller than G, without degrading too much the solution.

To solve these problems, we want:

- $\bar{S}_i \leq S_i^*$
- $\bar{S}_i = S_i^*$ for the "small" values of S_i^*

Thus, given a guess $G = (S_1^*, .., S_g^*)$:

- we choose a size j_1 bits for the significant, $j_1 \in \{1, .., \lceil \log(m) \rceil \}$
- we write $S_i^* = t_i 2^{x_i} + r_i$, with t_i encoded on j_1 bits, and $0 \leq x_i \leq \lceil \log(m) \rceil - j_1$, et $r_i \leq 2^{x_i} - 1$
- we define $\bar{S}_i = t_i 2^{x_i}$

We consider that the oracle gives $\bar{G}_2$. Notice that $|\bar{G}_2| = \Sigma_{i=1}^{g}(|t_i| + |x_i|) \leq g(j_1 + log(log(m))$.

Analysis of $MA^{\bar{G}_2}$

Proposition

$MA^{\bar{G}_2}$ is a $\beta + \frac{k-g-1}{g}$ approximation, with $1 + \frac{1}{2^{j_1-1}} = \beta$.

Proof:

- if $S_i^* \leq 2^{j_1} - 1$, then $\bar{S}_i = S_i^*$
- else, $\frac{S_i^*}{\bar{S}_i} = \frac{t_i 2^{x_i} + r_i}{t_i 2^{x_i}} \leq 1 + \frac{1}{t_i} \leq 1 + \frac{1}{2^{j_1-1}} = \beta$

Then, using the same analysis as MA^{G_2}:

$$\begin{aligned} T_{algo} &\leq \sum_{i=1}^{g} \beta T^*(h_i) + \sum_{i=g+1}^{k} \frac{S_i^*}{S_i} T^*(h_i) \\ &= \beta Opt + \underbrace{T^*(h_g)}_{\leq \frac{Opt}{g}}\left(\frac{m'}{a'} - k'\right) \end{aligned}$$

Summary

Outline of the derived approximation schemes:

algorithm	approximation ratio	complexity
MA^{G_1}	$(k-g)$	$O(m^g * kn)$
MA^{G_2}	$\frac{k-1}{g}$	$O((km)^g * kn)$
$MA^{\bar{G}_2}$	$\beta + \frac{k-g-1}{g}$	$O(k(2^{j_1} log(m))^g * kn)$

In [SFM06], k is fixed.

Conclusion

- Spatial heuristics combination
- Complexity vs. Approximation trade-off
- Partial oracle (To err is human !)

What's next ?

- real application experiments (SAT solvers)
- extend the method to other problems
- explore connections with PCP theory

Conclusion

- Spatial heuristics combination
- Complexity vs. Approximation trade-off
- Partial oracle (To err is human !)

What's next ?

- real application experiments (SAT solvers)
- extend the method to other problems
- explore connections with PCP theory

[SFM06] Tzur Sayag, Shai Fine, and Yishay Mansour.
Combining multiple heuristics.
2006.

[SW00] Petra Schuurman and Gerhard J. Woeginger.
Approximation schemes - a tutorial.
In *Lectures on Scheduling*, 2000.