Lien entre la conception de PTAS et les oracles (application au problème d'allocation de ressources dans un portfolio)

Marin Bougeret, Pierre-François Dutot, Alfredo Goldman, Yanik Ngoko, Denis Trystram

Laboratoire LIG

9 janvier 2009 Gotha MAO

1. Presentation of the problem

2. PTAS techniques and Oracle

3. Application of oracle techniques
   - First guess : arbitrary subset
   - Second guess : convenient subset
   - Guess approximation

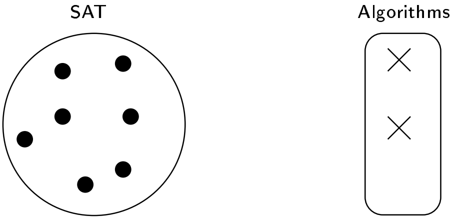
1. Presentation of the problem

2. PTAS techniques and Oracle

3. Application of oracle techniques
   - First guess : arbitrary subset
   - Second guess : convenient subset
   - Guess approximation

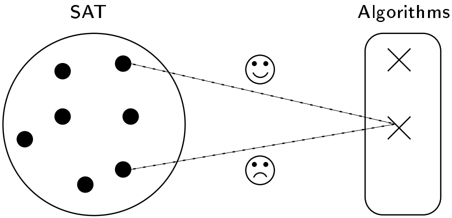
Introduction

- 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



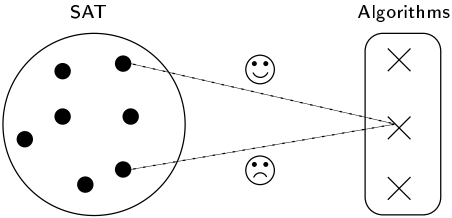
Introduction

- 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



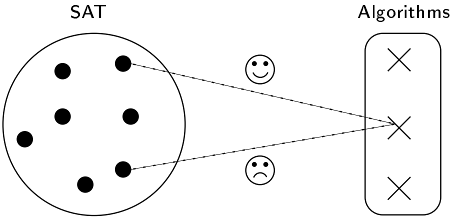
Introduction

- 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

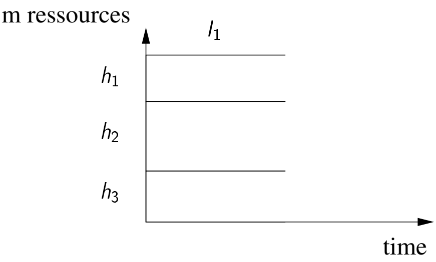
Outline :

- 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 every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using p ressources

Introduction

Outline :

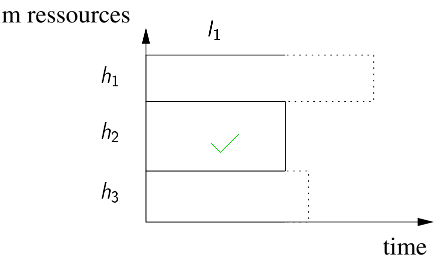
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using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



Introduction

Outline :

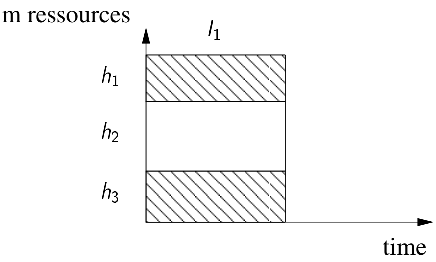
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using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



Introduction

Outline :

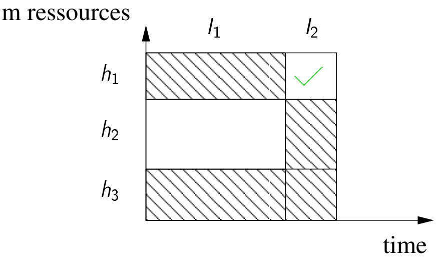
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using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



Introduction

Outline :

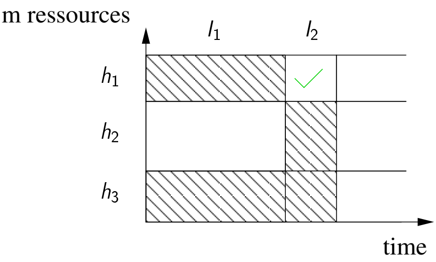
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using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



Introduction

Outline :

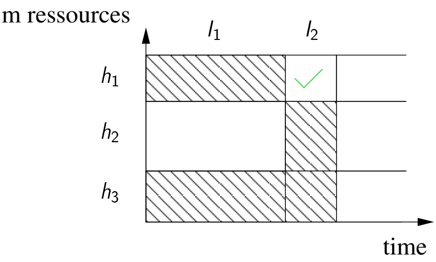
- a finite set of instances, a finite set of algorithm, a limited number of ressources m
- the goal is to minimize the total time to solve all the instances of the benchmark
- for every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using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

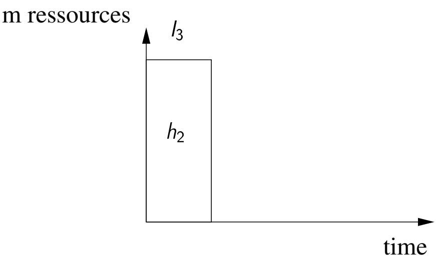


Introduction

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 every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using p ressources

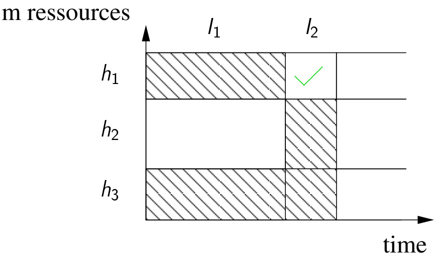


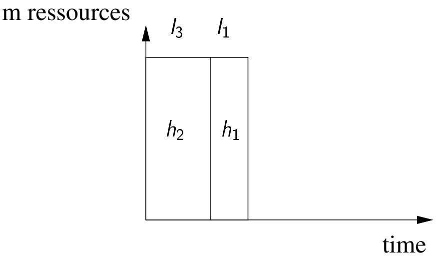


Introduction

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 every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using p ressources

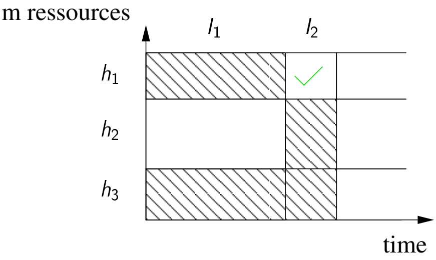


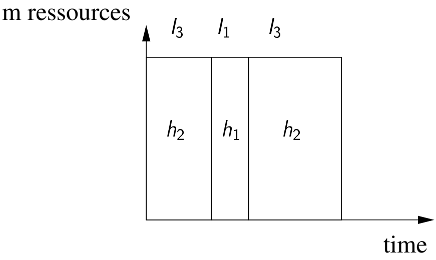


Introduction

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 every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using p ressources

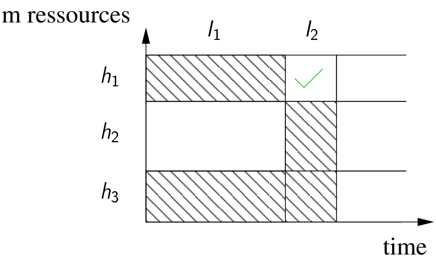


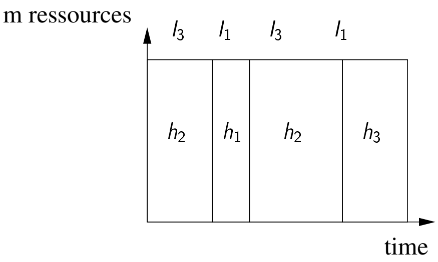


Introduction

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 every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using p ressources

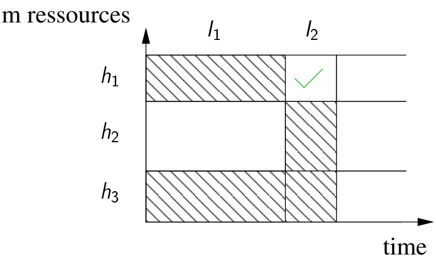


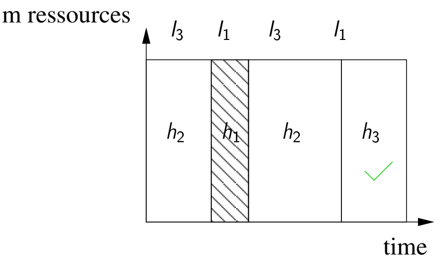


Introduction

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 every instance I_j, every algorithm h_i, every number of ressource p, to cost $C(h_i, I_j, p)$ for solving I_j with h_i using p ressources





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:

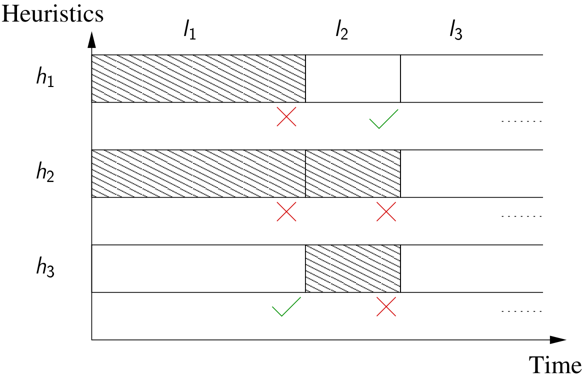
- a finite set of instances $I = \{I_1, \ldots, I_n\}$
- a finite set of heuristics $H = \{h_1, \ldots, h_k\}$
- m identical resources
- a cost $C(h_i, I_j, p) \in R^+$ for each $I_j \in I$, $h_i \in H$ and $p \in \{1, \ldots, m\}$

Continuous version ($p \in R^+$) in [2].

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 p the number of resources
- each heuristic must use at least one processor ($S_i \geq 1$), (well chosen portfolio)

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

NP hardness

The reduction is from the vertex cover problem. The input of the vertex cover problem is:

- k vertices
- n edges
- is there a vertex cover of size x ?

The input of the dRRSP is:

- k heuristics
- n instances in the benchmark
- x resources
- a cost matrix as follow (costs are indicated when using every resources)
- a threshold value T

	I_1	I_2	I_3	..	I_n
h_1	..	..	T+1	..	..
h_2	..	..	α	..	..
..	..	..	T+1	..	..
..	..	..	T+1	..	..
h_k	..	..	α	..	..

NP hardness

The input of the dRRSP is:

- k heuristics
- n instances in the benchmark
- x resources
- a cost matrix as follow (costs are indicated when using every resources)
- a threshold value T

	I_1	I_2	I_3	..	I_n
h_1	..	..	T+1	..	..
h_2	..	..	α	..	..
..	..	..	T+1	..	..
..	..	..	T+1	..	..
h_k	..	..	α	..	..

NP hardness

The input of the restricted dRRSP is:

- k heuristics
- n instances in the benchmark
- $x+k$ resources
- a cost matrix as follow (costs are indicated when using every resources)
- a threshold value T

	I_1	I_2	I_3	..	I_n
h_1	..	..	T+1	..	..
h_2	..	..	α	..	..
..	..	..	T+1	..	..
..	..	..	T+1	..	..
h_k	..	..	α	..	..

NP hardness

The input of the restricted dRRSP is:

- k heuristics
- $n + k$ instances in the benchmark
- $x + k$ resources
- a cost matrix as follow (costs are indicated when using every resources)
- a threshold value T

	I_1	I_2	I_3	..	I_n	I_{n+1}	I_{n+2}	..	..	I_{n+k}
h_1	..	..	$T+1$	..	..	Z	$T+1$	$T+1$	$T+1$	$T+1$
h_2	..	..	α	..	..	$T+1$	Z	$T+1$	$T+1$	$T+1$
..	..	..	$T+1$	..	..	$T+1$	$T+1$	Z	$T+1$	$T+1$
..	..	..	$T+1$	..	..	$T+1$	$T+1$	$T+1$	Z	$T+1$
h_k	..	..	α	..	..	$T+1$	$T+1$	$T+1$	$T+1$	Z

NP hardness

We will now choose T

- if there is a vertex cover of size x:
  $Opt \leq n\frac{\alpha m}{2} + Zm(k - x + \frac{x}{2}) = T$
- else, let's consider a solution S, and let $a = card\{S_i = 1\}$

NP hardness

If $a > k - x$:

$$\begin{aligned} Cost(S) &\geq Zm(a + \Sigma_{S_i \neq 1} \frac{1}{S_i}) \\ &= Zm(a + \Sigma_{S_i \neq 1} f(S_i)) \text{ with f convex} \\ &\geq Zm(a + (k-a) f(\frac{\Sigma_{S_i \neq 1} S_i}{k-a})) \\ &= Zm(a + \frac{(k-a)^2}{k+x-a}) \end{aligned}$$

And hence $Cost(S) - T \geq Zm(b) - \frac{n\alpha m}{2} > 0$, because $b > 0$ and Z can be chosen arbitrarily large.

If $a = k - x$:

$$\begin{aligned} Cost(S) &\geq (n-1)\frac{\alpha m}{2} + \alpha m + Zm(k - x + \frac{x}{2}) \\ &= T + \frac{\alpha m}{2} \end{aligned}$$

NP hardness

If $a > k - x$:

$$\begin{aligned} Cost(S) &\geq Zm(a + \Sigma_{S_i \neq 1} \frac{1}{S_i}) \\ &= Zm(a + \Sigma_{S_i \neq 1} f(S_i)) \text{ with f convex} \\ &\geq Zm(a + (k-a) f(\frac{\Sigma_{S_i \neq 1} S_i}{k-a})) \\ &= Zm(a + \frac{(k-a)^2}{k+x-a}) \end{aligned}$$

And hence $Cost(S) - T \geq Zm(b) - \frac{n\alpha m}{2} > 0$, because $b > 0$ and Z can be chosen arbitrarily large.
If $a = k - x$:

$$\begin{aligned} Cost(S) &\geq (n-1)\frac{\alpha m}{2} + \alpha m + Zm(k - x + \frac{x}{2}) \\ &= T + \frac{\alpha m}{2} \end{aligned}$$

A simple greedy algorithm

Notations (given a solution S):

- let $\sigma(j) = i_0 / \frac{C(h_{i_0}, I_j)}{S_{i_0}} = \min_{1 \le i \le k} \frac{C(h_i, I_j)}{S_i}$ 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

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

A simple greedy algorithm

Proposition

MA is a k approximation.

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

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

Introduction

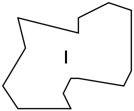
We present here some well known PTAS design techniques [3]:

- structuring the input
- structuring the output
- oracle based approach

Structuring the input

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

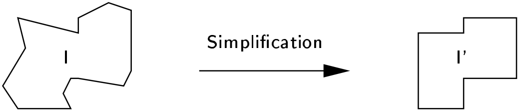
- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I



Structuring the input

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

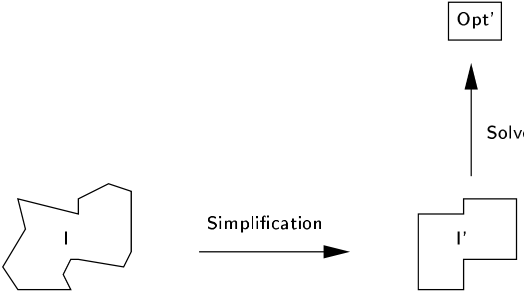
- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I



Structuring the input

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

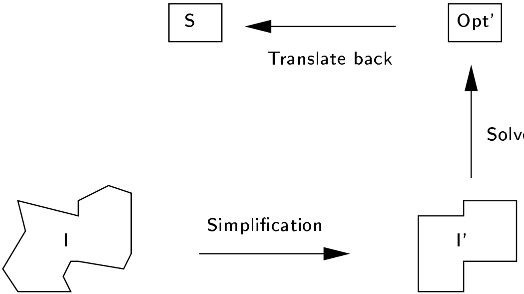
- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I



Structuring the input

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

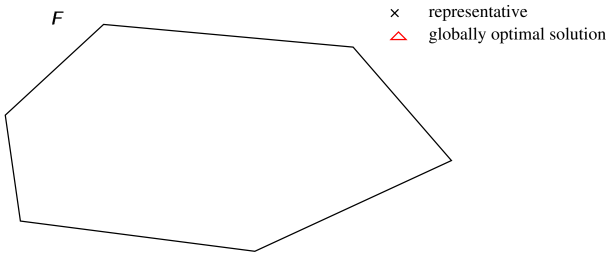
- **simplify**: turn I into a more primitive instance I' . This simplification depends on the desired precision ϵ
- **solve**: determine an optimal solution Opt' for I' (in polynomial time)
- **translate back**: translate the solution Opt' for I' into an approximate solution S for I



Structuring the output

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

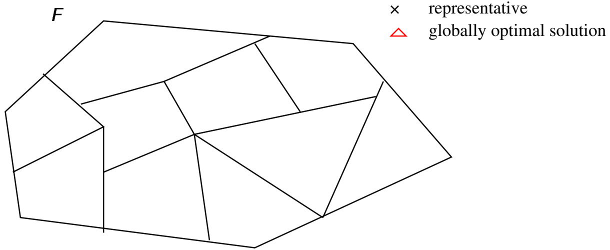
- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take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



Structuring the output

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

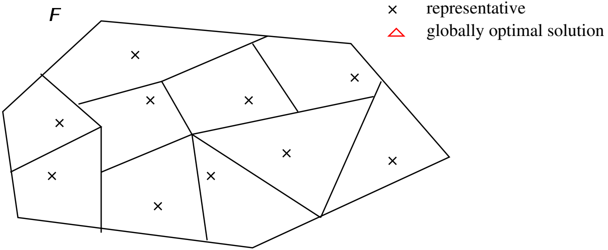
- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take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



Structuring the output

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

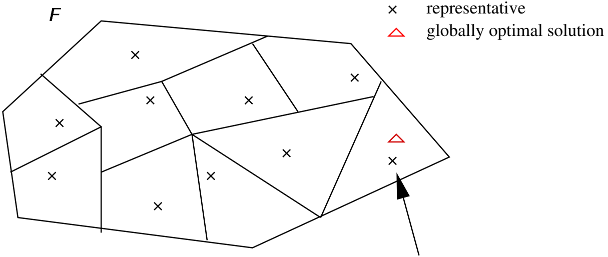
- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take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



Structuring the output

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

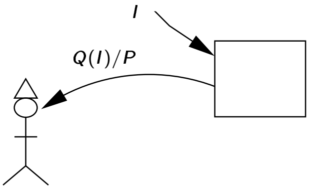
- **partition**: partition the feasible solution space F into a (polynomial) number of districts $F^{(1)}$, .., $F^{(d)}$. This partition depends on the desired precision ϵ.
- **find representative**: For each district $F^{(l)}$, determine a good representative $S^{(l)}$ "close" to $Opt^{(l)}$
- **take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



Oracle based approach

Another vision is possible, based on guesses from a reliable oracle.
Given in instance I, the main ("polynomial") steps are:

- **define the guess G**: choose a property P and ask a question $Q(I)$ to obtain it
- the oracle provides the approriate answer A of length L
- the guess is $G = Q(I) + A$
- **find a solution using the guess**: we get a solution $S(G, I)$
- **take the best**: try all the possible answers and select the best of all the $S(X, I)$

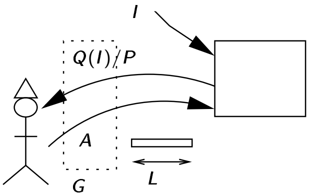


This technique seems equivalent to structuring the output, but ...

Oracle based approach

Another vision is possible, based on guesses from a reliable oracle. Given in instance I, the main ("polynomial") steps are:

- **define the guess G**: choose a property P and ask a question $Q(I)$ to obtain it
- the oracle provides the approriate answer A of length L
- the guess is $G = Q(I) + A$
- **find a solution using the guess**: we get a solution $S(G, I)$
- **take the best**: try all the possible answers and select the best of all the $S(X, I)$

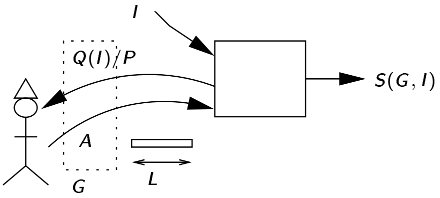


This technique seems equivalent to structuring the output, but ...

Oracle based approach

Another vision is possible, based on guesses from a reliable oracle. Given in instance I, the main ("polynomial") steps are:

- **define the guess G**: choose a property P and ask a question $Q(I)$ to obtain it
- the oracle provides the approriate answer A of length L
- the guess is $G = Q(I) + A$
- **find a solution using the guess**: we get a solution $S(G, I)$
- **take the best**: try all the possible answers and select the best of all the $S(X, I)$

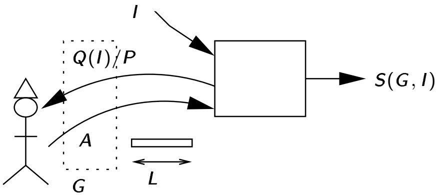


This technique seems equivalent to structuring the output, but ...

Oracle based approach

Another vision is possible, based on guesses from a reliable oracle. Given in instance I, the main ("polynomial") steps are:

- **define the guess G**: choose a property P and ask a question $Q(I)$ to obtain it
- the oracle provides the approriate answer A of length L
- the guess is $G = Q(I) + A$
- **find a solution using the guess**: we get a solution $S(G, I)$
- **take the best**: try all the possible answers and select the best of all the $S(X, I)$

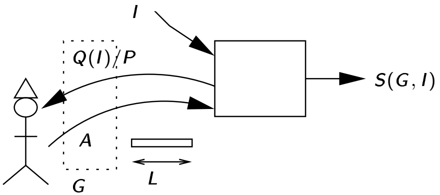


This technique seems equivalent to structuring the output, but ...

Oracle based approach

Another vision is possible, based on guesses from a reliable oracle. Given in instance I, the main ("polynomial") steps are:

- **define the guess G**: choose a property P and ask a question $Q(I)$ to obtain it
- the oracle provides the approriate answer A of length L
- the guess is $G = Q(I) + A$
- **find a solution using the guess**: we get a solution $S(G, I)$
- **take the best**: try all the possible answers and select the best of all the $S(X, I)$



This technique seems equivalent to structuring the output, but ..

The oracle based approach (2)

What means G, and how using it ?

- G represents a constraint on the problem variables. Respecting G ensures that P is true.
- the solution $S(G, I)$ does not necessarily respect the constraint G

Moreover, the oracle based approach leads to another technique..

The oracle based approach (3) : guess approximation

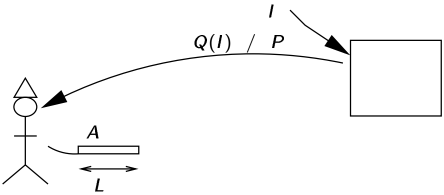
A natural idea is to look for a compact way for expressing G.

- idea(1): outline approximation schemes = structuring the input + giving a guess [1]
- idea(2): guess approximation = approximate the guess itself !
- idea(3): .. ?

The oracle based approach (3) : guess approximation

A natural idea is to look for a compact way for expressing G.

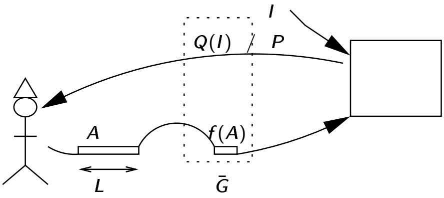
- idea(1): outline approximation schemes = structuring the input + giving a guess [1]
- idea(2): guess approximation = approximate the guess itself !
- idea(3): .. ?



The oracle based approach (3) : guess approximation

A natural idea is to look for a compact way for expressing G.

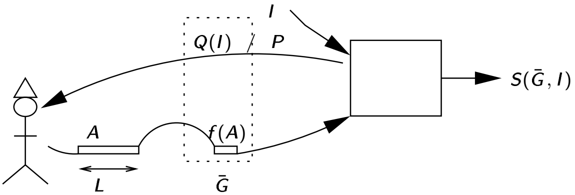
- idea(1): outline approximation schemes = structuring the input + giving a guess [1]
- idea(2): guess approximation = approximate the guess itself !
- idea(3): .. ?



The oracle based approach (3) : guess approximation

A natural idea is to look for a compact way for expressing G.

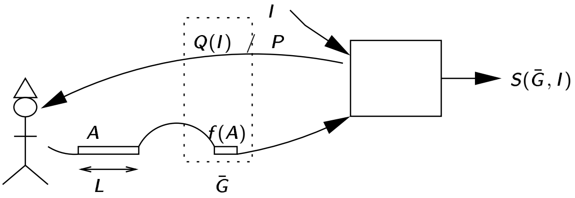
- idea(1): outline approximation schemes = structuring the input + giving a guess [1]
- idea(2): guess approximation = approximate the guess itself !
- idea(3): .. ?



The oracle based approach (3) : guess approximation

A natural idea is to look for a compact way for expressing G.

- idea(1): outline approximation schemes = structuring the input + giving a guess [1]
- idea(2): guess approximation = approximate the guess itself !
- idea(3): .. ?



Guess 1

As a first step, we choose arbitrarily g heuristics denoted by $\{h_1, \ldots, h_g\}$.

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 choose arbitrarily g heuristics denoted by $\{h_1, \ldots, h_g\}$.

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 $m' = a'k' + b', b' < k'$

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

Algorithm MA_R^G

Remarks:

- MA_R^G doesn't respect G
- MA_R^G requires an $s > k + c$.. a solution to ensure this is to ask a stronger property P : there exists an optimal solution such that
  - S_i^* processors are allocated to $h_i, i \in \{1, .., g\}$
  - $\exists i_0 \in \{1, .., g\}$ such that $S_{i_0}^* \geq S_i, i \in \{1, .., k\}$

Thus, we need a larger guess to indicates the index i_0.
We will now look for stronger properties., *ie* we no longer choose an arbitrary subset of heuristics.

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.

Insight:

- assume that we choose $\bar{G}$ such that $S_i^* = \bar{S}_i \pm 1, \forall i \in \{1, .., g\}$
- then, one guess cover 3^g possibilities

Problems:

- for the guessed heuristics, we don't know if we are suboptimal or over-optimal
- $\bar{S}_i = S_i^* - 1$ is very bad if S_i^* is small
- if $\Sigma_{i=1}^{g} \bar{S}_i > \Sigma_{i=1}^{g} \bar{S}_i^*$, we may have less remaining processors when applying MA

Introduction

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

Insight:

- assume that we choose $\bar{G}$ such that $S_i^* = \bar{S}_i \pm 1, \forall i \in \{1, .., g\}$
- then, one guess cover 3^g possibilities

Problems:

- for the guessed heuristics, we don't know if we are suboptimal or over-optimal
- $\bar{S}_i = S_i^* - 1$ is very bad if S_i^* is small
- if $\Sigma_{i=1}^{g} \bar{S}_i > \Sigma_{i=1}^{g} \bar{S}_i^*$, we may have less remaining processors when applying MA

Definition of $\bar{G}$

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}:

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

Outline of the part

Outline of the derived PTASs:

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

Conclusion

In this presentation:

- we extended the resource sharing problem to the discrete version (dRRSP)
- we proved the NP hardness of the restricted version we are interested in
- we presented an small overview of PTAS's techniques, and introduced the guess approximation methodology
- we applied this methodology on the dRRSP

Conclusion

In this presentation:

- we extended the resource sharing problem to the discrete version (dRRSP)
- we proved the NP hardness of the restricted version we are interested in
- we presented an small overview of PTAS's techniques, and introduced the guess approximation methodology
- we applied this methodology on the dRRSP

Bibliography

[1] L. A. Hall and D. B. Shmoys.
Approximation schemes for constrained scheduling problems.
pages 134–139, 1989.

[2] T. Sayag, S. Fine, and Y. Mansour.
Combining multiple heuristics.
2006.

[3] P. Schuurman and G. J. Woeginger.
Approximation schemes - a tutorial.
In *Lectures on Scheduling*, 2000.