Combining Multiple Heuristics on Discrete Resources

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

LIG laboratory, France

May 25th, APDCM 09

1. Presentation of the problem

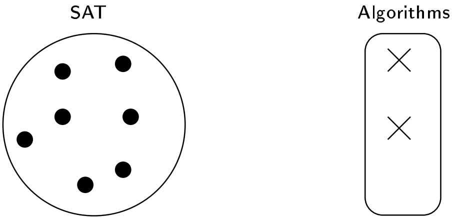
2. Oracle formalism for PTAS design

3. Approximation schemes for the restricted dRSSP
   - First guess : arbitrary subset
   - Second guess : convenient subset

Introduction

Context:

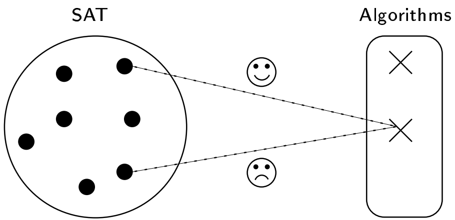
- a finite set H of algorithms/heuristics (algorithm portfolio)
- a finite set I of "representatives" instances (benchmark)
- the time needed by each algorithm of H to solve each instance of I is known
- goal : minimize the time needed to solve all the instances from the benchmark
- more than selection : combination of algorithms



Introduction

Context:

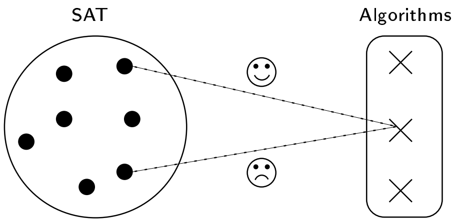
- a finite set H of algorithms/heuristics (algorithm portfolio)
- a finite set I of "representatives" instances (benchmark)
- the time needed by each algorithm of H to solve each instance of I is known
- goal : minimize the time needed to solve all the instances from the benchmark
- more than selection : combination of algorithms



Introduction

Context:

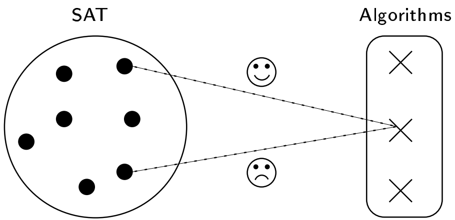
- a finite set H of algorithms/heuristics (algorithm portfolio)
- a finite set I of "representatives" instances (benchmark)
- the time needed by each algorithm of H to solve each instance of I is known
- goal : minimize the time needed to solve all the instances from the benchmark
- more than selection : combination of algorithms



Introduction

Context:

- a finite set H of algorithms/heuristics (algorithm portfolio)
- a finite set I of “representatives” instances (benchmark)
- the time needed by each algorithm of H to solve each instance of I is known
- goal : minimize the time needed to solve all the instances from the benchmark
- more than selection : combination of algorithms



Introduction

What we mean by combination :

- one instance may be solved by several algorithms in parallel
- parallel task model : moldable
- when a solution of an instance is found, everyone is aware

Introduction

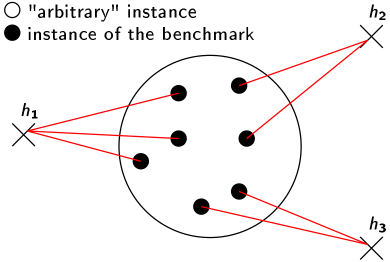
Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?

Introduction

Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!

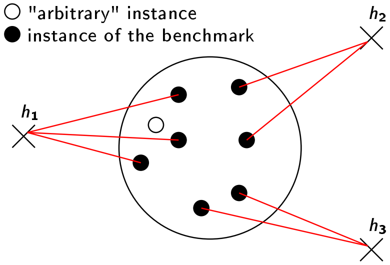
Introduction

Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!



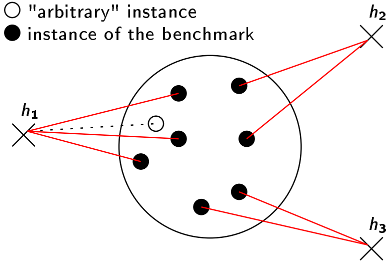
Introduction

Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!



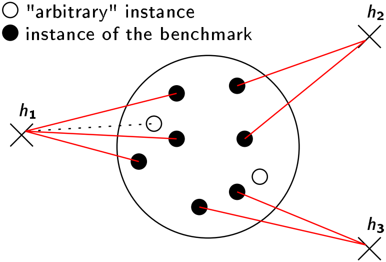
Introduction

Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!



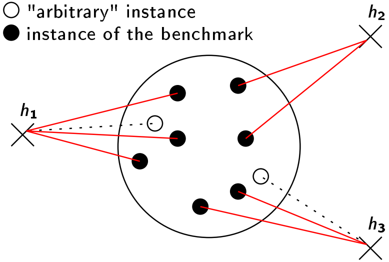
Introduction

Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!



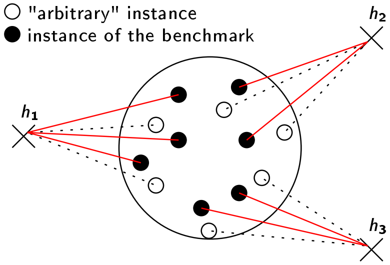
Introduction

Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!



Introduction

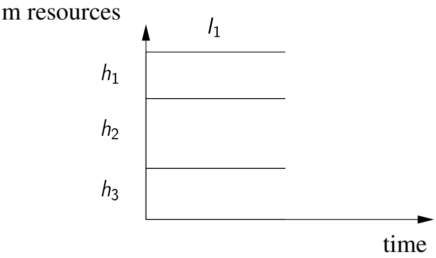
Why not choosing the following greedy optimal policy: for each instance, give all the resources to the best heuristic ?
Because we don't want to solve ONLY the benchmark!



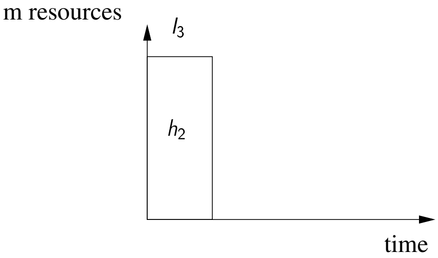
Introduction

Some existing combination models:

Space sharing [1]



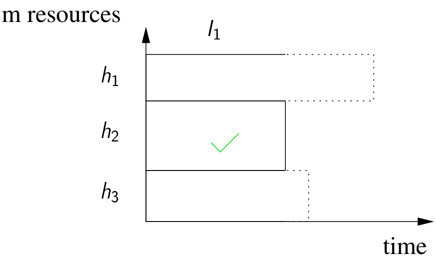
Time sharing [1]



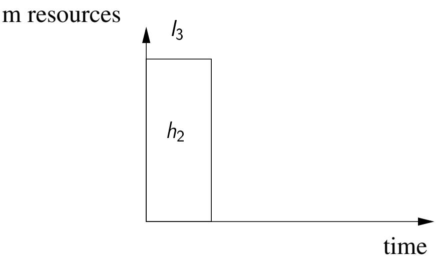
Introduction

Some existing combination models:

Space sharing [1]



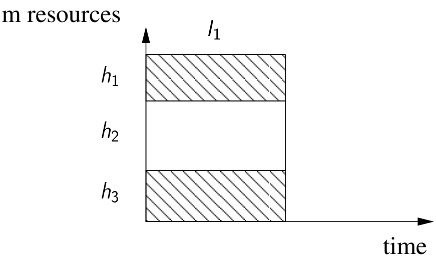
Time sharing [1]



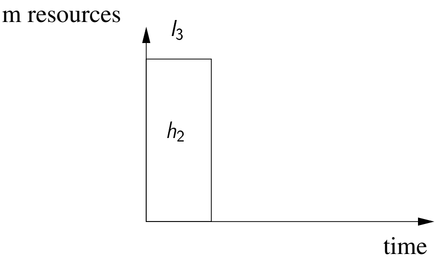
Introduction

Some existing combination models:

Space sharing [1]



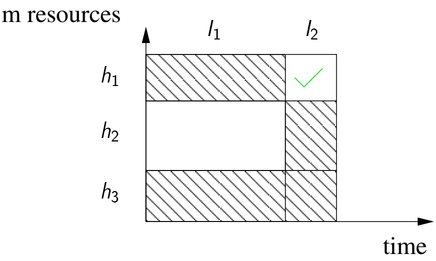
Time sharing [1]



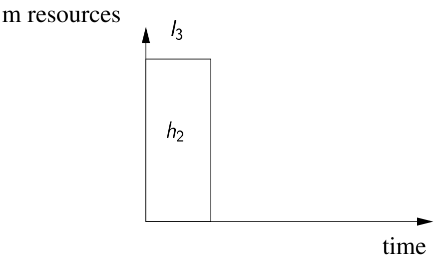
Introduction

Some existing combination models:

Space sharing [1]



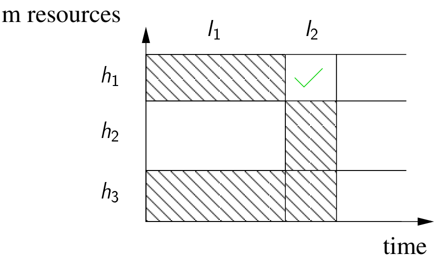
Time sharing [1]



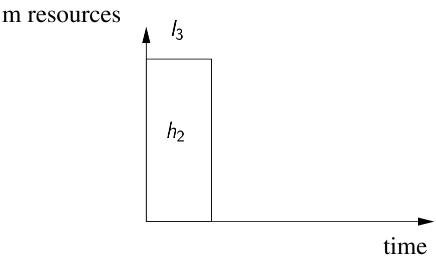
Introduction

Some existing combination models:

Space sharing [1]



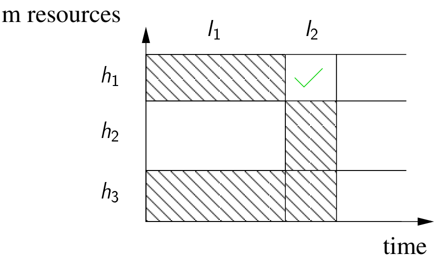
Time sharing [1]



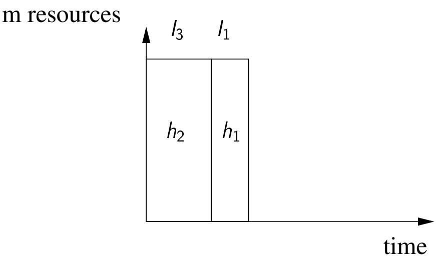
Introduction

Some existing combination models:

Space sharing [1]



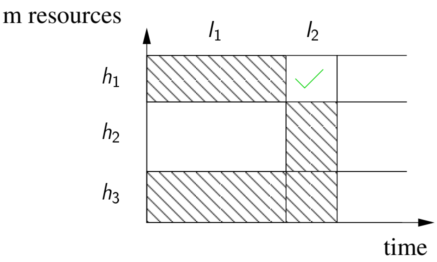
Time sharing [1]



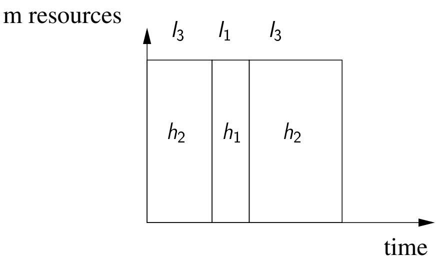
Introduction

Some existing combination models:

Space sharing [1]



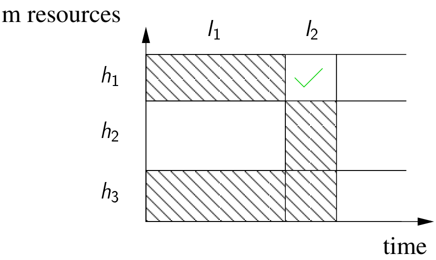
Time sharing [1]



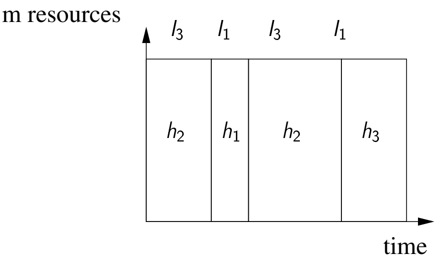
Introduction

Some existing combination models:

Space sharing [1]



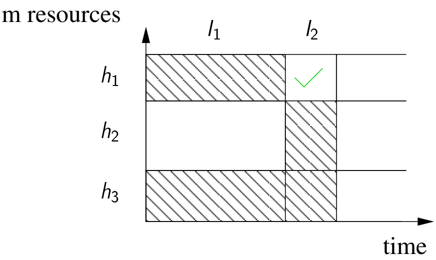
Time sharing [1]



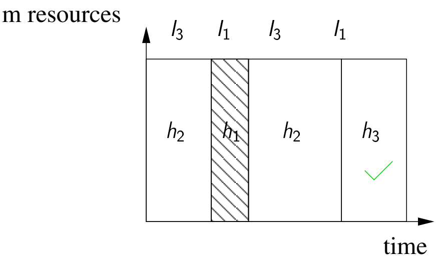
Introduction

Some existing combination models:

Space sharing [1]



Time sharing [1]



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 $p \in \{1, \ldots, m\}$

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

- $S_i \in \mathbb{N}$
- $\sum_{i=1}^{k} S_i = m$
- S minimizes $\sum_{j=1}^{n} \min_{1 \le i \le k} \{C(h_i, I_j, S_i)\}$

Continuous version ($p \in R^+$) in [1].

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 $p \in \{1, \ldots, m\}$

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

- $S_i \in \mathbb{N}$
- $\sum_{i=1}^{k} S_i = m$
- S minimizes $\sum_{j=1}^{n} \min_{1 \le i \le k} \{C(h_i, I_j, S_i)\}$

Continuous version ($p \in R^+$) in [1].

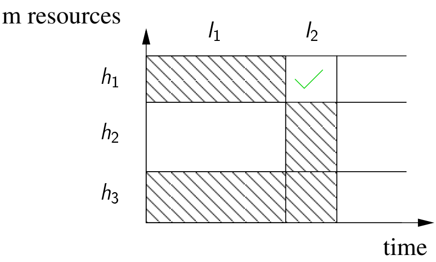
Complexity results

- the dRSSP is NP hard in the strong sense (reduction from the vertex cover problem)
- the dRSSP is innaproximable (unless $P = NP$) within a constant factor (if $m < k$)
- we study a restricted version:
  - **linear cost assumption:** $C(h_i, I_j, p) = C(h_i, I_j, m)\frac{m}{p}$
  - **well chosen portfolio:** each heuristic must use at least one processor ($S_i \geq 1$)

Notations

Given a solution S:

- let $\sigma(j) = \underset{1 \leq i \leq k}{argmin} \frac{C(h_i, I_j)}{S_i}$ 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
- let $T(h_i) = \Sigma_{j/\sigma(j)=i} T(I_j)$ is the "useful" computation time of heuristic i in S



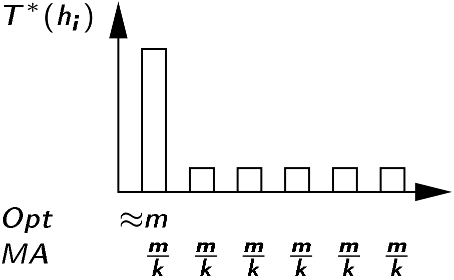
A naive algorithm : MA

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.

Proof/Worst case:



1. Presentation of the problem
2. Oracle formalism for PTAS design
3. Approximation schemes for the restricted dRSSP
   - First guess : arbitrary subset
   - Second guess : convenient subset

Introduction

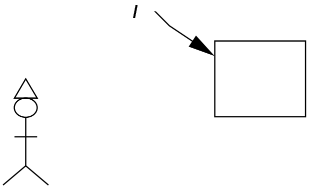
Some of the main *PTAS* design techniques [2]:

- structuring the input
- structuring the output ("extending partial small size solutions")
- structuring the execution of an algorithm ("trimmed algorithm")
- rounding *LP*
- oracle based approach
- ...

Oracle based approach

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

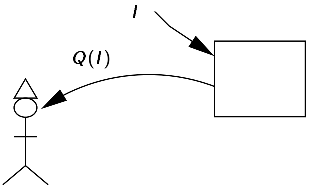
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: an algorithm A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

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

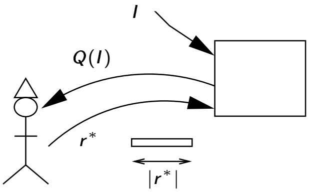
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: an algorithm A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

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

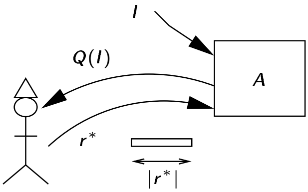
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: an algorithm A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

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

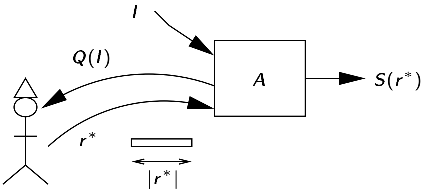
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: an algorithm A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

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

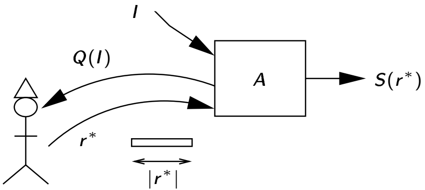
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: an algorithm A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best of all the $S(r), r \in R$



Oracle based approach

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

- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: an algorithm A provides $S(r^*) \leq \rho Opt$
- take the best: 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 $|r^*|$ (leading to different ρ), leading to classical approximation schemes.

1. Presentation of the problem

2. Oracle formalism for PTAS design

3. Approximation schemes for the restricted dRSSP
   - First guess : arbitrary subset
   - Second guess : convenient subset

Introduction

In this part:

- we use the *MA* algorithm as a basis
- we look for “the right” question to ask to the oracle

Introduction

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 guesses from the oracle.

Guess 1

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

Definition G1

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 use the algorithm MA^G with $G = G_1$.

Proposition

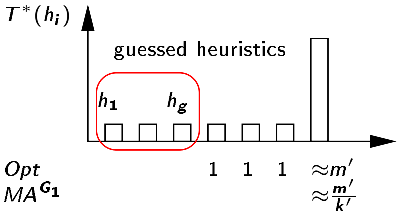
MA^{G_1} is a $k - g$ approximation.

Analysis of MA^{G_1}

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

Proof/Worst case:



Algorithm MA_R^G

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

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

Proposition

With the "right" α, $MA_R^{G_1}$ is a $(k-g)(1-\beta)$ approximation

MA_R^G requires a larger guess to ensure that $s > k + c$ (c constant)

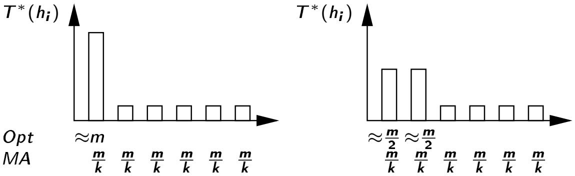
Introduction

The oracle response $r^* = [(i_1^*, \ldots, i_g^*), (r_1^*, \ldots, r_g^*)]$ will indicate the number of resources r_j^* allocated to heuritic $h_{i_j^*}$ (in a optimal solution).
The question now is: given an optimal solution S^*, what is the most "important" subset of g heuristics ?

- 1) those that have the largest number of allocated resources
- 2) those that have the fewest number of allocated resources
- 3) those that have the largest "useful" computation time

Another analysis of MA

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



Difficult instances are the ones where the optimal only uses a small number of heuristics.

Another analysis of MA

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

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^*.

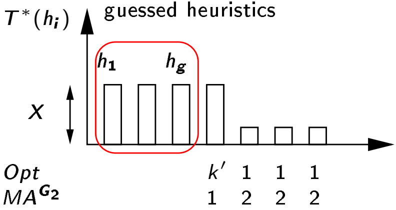
Notice that $|G_2| = glog(k) + glog(m)$.
We use the algorithm MA^G with $G = G_2$.

Proposition

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

Analysis of MA^{G_2}

Proof/Worst case:



$$Opt \approx (g+1)X$$
$$MA^{G_2} \approx gX + k'X = kX$$

Outline of the obtained approximation schemes

algorithm	approx. ratio	$\|r^*\|$	complexity
MA^{G_1}	$(k-g)$	$glog(m)$	$O(m^g * kn)$
$MA_R^{G_1}$	$(k-g)(1-\beta)$	$log(k) + glog(m)$	$O(k * m^g * kn)$
MA^{G_2}	$\frac{k}{g+1}$	$g(log(k) + log(m))$	$O((km)^g * kn)$

Remark: these results are approximation schemes, and not $PTAS$

Conclusion

In this presentation:

- we extended the resource sharing problem to the discrete version (dRSSP)
- we presented the oracle methodology for PTAS design
- we built different approximation schemes for the restricted dRSSP

Conclusion

In this presentation:

- we extended the resource sharing problem to the discrete version (dRSSP)
- we presented the oracle methodology for PTAS design
- we built different approximation schemes for the restricted dRSSP

Bibliography

[1] T. Sayag, S. Fine, and Y. Mansour.
Combining multiple heuristics.
In *STACS 2006*, volume 3884, pages 242–253. Springer, 2006.

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

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 ?

NP hardness

The input of the dRSSP is:

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

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

How to choose the right value for T:

- if there exists a vertex cover of size x, then $opt \leq nx\alpha = T$
- else $opt \geq T + 1$

The gap can be arbitrary large.

The reduction for the restricted version is based on the same idea.