The guess approximation technique and its application to the Discrete Resource Sharing Scheduling Problem

Marin Bougeret, Pierre-François Dutot, Denis Trystram

Laboratoire LIG

MAPSP 2009

1. Brief overview of classical *PTAS* design techniques
2. The guess approximation technique
3. Application to the *DRSSP*
    - Presentation of the problem
    - A first approximation scheme
    - Improved scheme with guess approximation
4. Discussion

1. Brief overview of classical *PTAS* design techniques
2. The guess approximation technique
3. Application to the *DRSSP*
   - Presentation of the problem
   - A first approximation scheme
   - Improved scheme with guess approximation
4. Discussion

The main techniques..

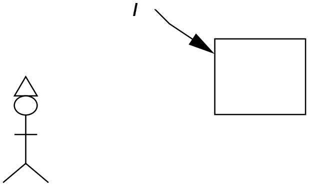
Some of the main classical *PTAS* design techniques [3] [4]:

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

Oracle based approach

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

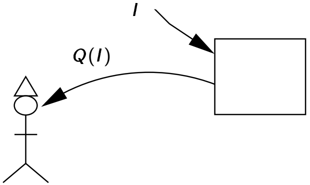
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best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



Oracle based approach

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

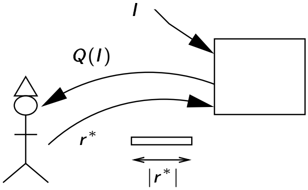
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best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



Oracle based approach

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

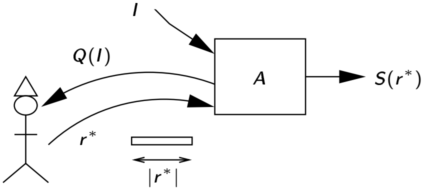
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best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



Oracle based approach

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

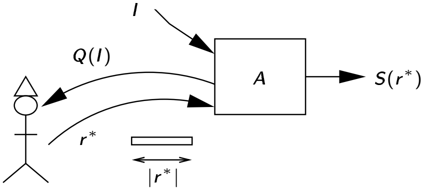
- define the guess G: choose an "interesting" property P
- ask a question $Q(I)$ to the oracle
- the oracle provides an answer $r^* \in R$ (s t. $P(Q(I), r^*)$ is true)
- find a solution using the guess: A provides $S(r^*) \leq \rho Opt$
- take the best: try all the possible answers and select the best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



Oracle based approach

This technique is based on guesses from a reliable oracle. Given in 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: 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

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. Brief overview of classical *PTAS* design techniques
2. The guess approximation technique
3. Application to the *DRSSP*
   - Presentation of the problem
   - A first approximation scheme
   - Improved scheme with guess approximation
4. Discussion

Definition

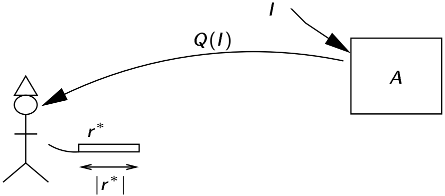
A natural idea is to look for a compact way for expressing the oracle answer.

- idea(1): outline approximation schemes = structuring the input + asking question [1]
- idea(2): guess approximation = approximate the guess itself !

Definition

A natural idea is to look for a compact way for expressing the oracle answer.

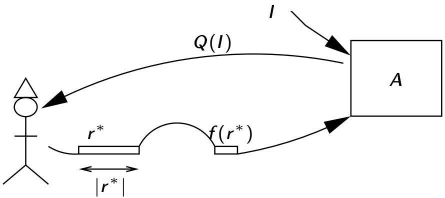
- idea(1): outline approximation schemes = structuring the input + asking question [1]
- idea(2): guess approximation = approximate the guess itself !



Definition

A natural idea is to look for a compact way for expressing the oracle answer.

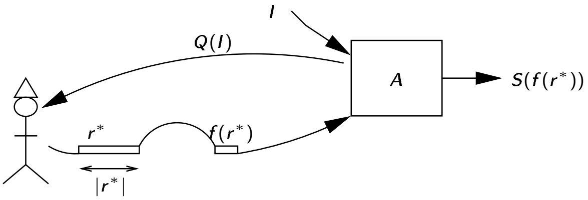
- idea(1): outline approximation schemes = structuring the input + asking question [1]
- idea(2): guess approximation = approximate the guess itself !



Definition

A natural idea is to look for a compact way for expressing the oracle answer.

- idea(1): outline approximation schemes = structuring the input + asking question [1]
- idea(2): guess approximation = approximate the guess itself !



Consequences

The obtained algorithm (without oracle):

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

Here, we can control the complexity by adjusting:

- the length of the needed oracle answer
- the roughness of the contraction

1. Brief overview of classical *PTAS* design techniques
2. The guess approximation technique
3. Application to the *DRSSP*
   - Presentation of the problem
   - A first approximation scheme
   - Improved scheme with guess approximation
4. Discussion

Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Input

- a set of n instances I_j, a set of k heuristics h_i, m resources to share
- a cost matrix (C_{ij}) which gives the time needed for any heuristic h_i to solve any instance I_j with 1 resource (+ linear assumption)

Output

An allocation of the resources $S = (S_1, \ldots, S_k)$ such that:

- $S_i \in \mathbb{N}^*$ (continuous version in [2])
- $\sum_{i=1}^{k} S_i = m$

Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Input

- a set of n instances I_j, a set of k heuristics h_i, m resources to share
- a cost matrix (C_{ij}) which gives the time needed for any heuristic h_i to solve any instance I_j with 1 resource (+ linear assumption)

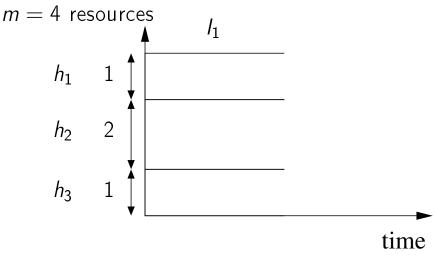
Output

An allocation of the resources $S = (S_1, \ldots, S_k)$ such that:

- $S_i \in \mathbb{N}^*$ (continuous version in [2])
- $\sum_{i=1}^{k} S_i = m$

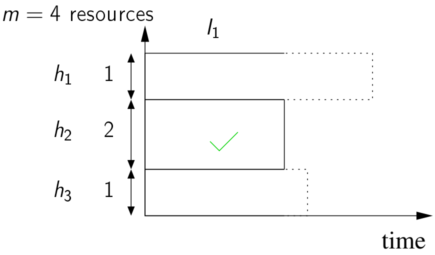
Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Objective function: $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{ \frac{C(h_i, I_j)}{S_i} \}$



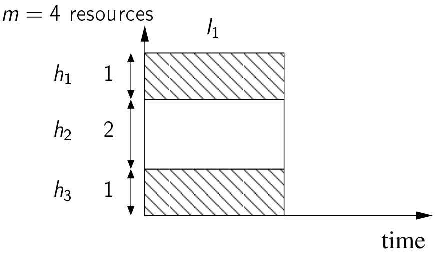
Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Objective function: $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{ \frac{C(h_i, l_j)}{S_i} \}$



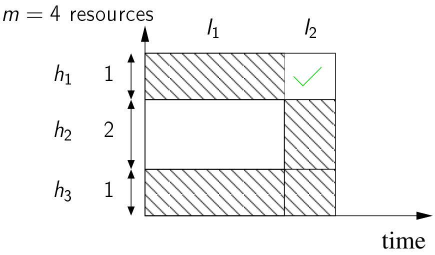
Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Objective function: $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{ \frac{C(h_i, I_j)}{S_i} \}$



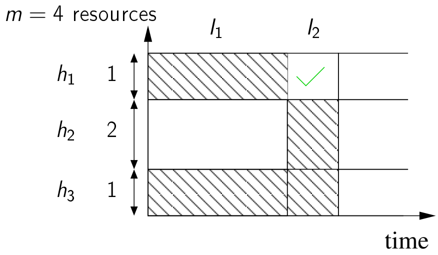
Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Objective function: $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{ \frac{C(h_i, I_j)}{S_i} \}$



Discrete Resource Sharing Scheduling Problem (*DRSSP*)

Objective function: $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{ \frac{C(h_i, I_j)}{S_i} \}$



Definition of the algorithm

MA : the "core" algorithm

- mean allocation algorithm (*MA*): allocates $\lfloor \frac{m}{k} \rfloor$ resources to each heuristic.
- *MA* is a k approximation.

MA with oracle : MA^r

- we choose $g \in \{1, \ldots, k\}$, which parameterizes the length of the oracle response
- we consider the following MA^r algorithm (given any guess $r = [(i_1, \ldots, i_g), (r_1, \ldots, r_g)]$):
  - allocate r_j processors to heuristic $h_{i_j}, j \in \{1, \ldots, g\}$
  - applies *MA* on the k' others heuristics with the m' remaining processors

Definition of the algorithm

MA : the "core" algorithm

- mean allocation algorithm (MA): allocates $\lfloor \frac{m}{k} \rfloor$ resources to each heuristic.
- MA is a k approximation.

MA with oracle : MA^r

- we choose $g \in \{1, \ldots, k\}$, which parameterizes the length of the oracle response
- we consider the following MA^r algorithm (given any guess $r = [(i_1, \ldots, i_g), (r_1, \ldots, r_g)]$):
  - allocate r_j processors to heuristic $h_{i_j}, j \in \{1, \ldots, g\}$
  - applies MA on the k' others heuristics with the m' remaining processors

MA^r with the "good" property

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

Proposition

When asking to the oracle the allocation of the g heuristics which verify property 3

- MA^r is a $\frac{k}{g+1}$ approximation
- complexity of $MA^r \approx (km)^g$

MA^r with the "good" property

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

Proposition

When asking to the oracle the allocation of the g heuristics which verify property 3

- MA^r is a $\frac{k}{g+1}$ approximation
- complexity of $MA^r \approx (km)^g$

What could be approximated here?

Notice that the oracle provides two types of information:

- a set of index of heuristics (hard to "contract")
- a set of number of allocated processors (easy to "contract")

We need to define f such that

- $|f(r^*)| << |r^*|$
- the approximation ratio using $f(r^*)$ is not degraded too much

Thus we contract the vector $(r_1^*, \ldots, r_g^*)$.
Let $(\tilde{r_1}^*, \ldots, \tilde{r_g}^*) = f(r_1^*, \ldots, r_g^*)$.

What could be approximated here?

Notice that the oracle provides two types of information:

- a set of index of heuristics (hard to “contract”)
- a set of number of allocated processors (easy to “contract”)

We need to define f such that

- $|f(r^*)| << |r^*|$
- the approximation ratio using $f(r^*)$ is not degraded too much

Thus we contract the vector $(r_1^*, \ldots, r_g^*)$.
Let $(\tilde{r_1}^*, \ldots, \tilde{r_g}^*) = f(r_1^*, \ldots, r_g^*)$.

What could be approximated here?

Notice that the oracle provides two types of information:

- a set of index of heuristics (hard to “contract”)
- a set of number of allocated processors (easy to “contract”)

We need to define f such that

- $|f(r^*)| << |r^*|$
- the approximation ratio using $f(r^*)$ is not degraded too much

Thus we contract the vector $(r_1^*, \ldots, r_g^*)$.
Let $(\tilde{r_1}^*, \ldots, \tilde{r_g}^*) = f(r_1^*, \ldots, r_g^*)$.

What could be approximated here?

Notice that the oracle provides two types of information:

- a set of index of heuristics (hard to "contract")
- a set of number of allocated processors (easy to "contract")

We need to define f such that

- $|f(r^*)| << |r^*|$
- the approximation ratio using $f(r^*)$ is not degraded too much

Thus we contract the vector $(r_1^*, \ldots, r_g^*)$.
Let $(\tilde{r_1}^*, \ldots, \tilde{r_g}^*) = f(r_1^*, \ldots, r_g^*)$.

How defining the $\tilde{r_i}^*$?

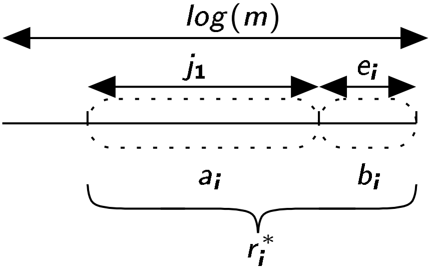
We need:

- $\tilde{r_i}^* \leq r_i^*$
- if r_i^* is small, we must have $\tilde{r_i}^* \approx r_i^*$

Thus, we only keep the j_1 most significant bit of the r_i^*.

- let $r_i^* = a_i 2^{e_i} + b_i$
- we define $\tilde{r_i}^* = a_i 2^{e_i}$

Then, $|\tilde{r_i}^*| = log(a_i) + log(e_i) \leq j_1 + log(log(m))$.



How defining the $\tilde{r}_i^*$?

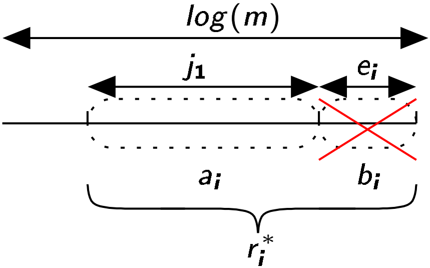
We need:

- $\tilde{r}_i^* \leq r_i^*$
- if r_i^* is small, we must have $\tilde{r}_i^* \approx r_i^*$

Thus, we only keep the j_1 most significant bit of the r_i^*.

- let $r_i^* = a_i 2^{e_i} + b_i$
- we define $\tilde{r}_i^* = a_i 2^{e_i}$

Then, $|\tilde{r}_i^*| = log(a_i) + log(e_i) \leq j_1 + log(log(m))$.



New ratio using $f(r^*)$

The ratio is directly linked to $\frac{r_i^*}{\tilde{r}_i}$.
For a given guessed heuristic i:

- if $r_i^* \leq 2^{j_1} - 1, \tilde{r}_i^* = r_i$
- if $r_i^* \geq 2^{j_1}, \frac{r_i^*}{\tilde{r}_i^*} = \frac{a_i 2^{e_i} + b_i}{a_i 2^{e_i}} = 1 + \frac{2^{e_i}}{a_i 2^{e_i}} \leq 1 + \frac{1}{2^{j_1 - 1}} = \beta$

Proposition

When using the guess approximation technique:

- MA^r is a $\beta + \frac{k-g}{g+1}(2-\beta) \leq \beta + \frac{k-g}{g+1}$ approximation
- new complexity of $MA^r \approx (k 2^{j_1} log(m))^g$ (Versus $(km)^g$)

New ratio using $f(r^*)$

The ratio is directly linked to $\frac{r_i^*}{\tilde{r}_i}$.
For a given guessed heuristic i:

- if $r_i^* \leq 2^{j_1} - 1$, $\tilde{r}_i{}^* = r_i$
- if $r_i^* \geq 2^{j_1}$, $\frac{r_i^*}{\tilde{r}_i{}^*} = \frac{a_i 2^{e_i} + b_i}{a_i 2^{e_i}} = 1 + \frac{2^{e_i}}{a_i 2^{e_i}} \leq 1 + \frac{1}{2^{j_1 - 1}} = \beta$

Proposition

When using the guess approximation technique:

- MA^r is a $\beta + \frac{k-g}{g+1}(2 - \beta) \leq \beta + \frac{k-g}{g+1}$ approximation
- new complexity of $MA^r \approx (k 2^{j_1} log(m))^g$ (Versus $(km)^g$)

New ratio using $f(r^*)$

The ratio is directly linked to $\frac{r_i^*}{\tilde{r}_i}$.
For a given guessed heuristic i:

- if $r_i^* \leq 2^{j_1} - 1$, $\tilde{r_i}^* = r_i$
- if $r_i^* \geq 2^{j_1}$, $\frac{r_i^*}{\tilde{r_i}^*} = \frac{a_i 2^{e_i} + b_i}{a_i 2^{e_i}} = 1 + \frac{2^{e_i}}{a_i 2^{e_i}} \leq 1 + \frac{1}{2^{j_1 - 1}} = \beta$

Proposition

When using the guess approximation technique:

- MA^r is a $\beta + \frac{k-g}{g+1}(2-\beta) \leq \beta + \frac{k-g}{g+1}$ approximation
- new complexity of $MA^r \approx (k 2^{j_1} log(m))^g$ (Versus $(km)^g$)

New ratio using $f(r^*)$

The ratio is directly linked to $\frac{r_i^*}{\tilde{r}_i}$.
For a given guessed heuristic i:

- if $r_i^* \leq 2^{j_1} - 1$, $\tilde{r}_i^* = r_i$
- if $r_i^* \geq 2^{j_1}$, $\frac{r_i^*}{\tilde{r}_i^*} = \frac{a_i 2^{e_i} + b_i}{a_i 2^{e_i}} = 1 + \frac{2^{e_i}}{a_i 2^{e_i}} \leq 1 + \frac{1}{2^{j_1 - 1}} = \beta$

Proposition

When using the guess approximation technique:

- MA^r is a $\beta + \frac{k-g}{g+1}(2-\beta) \leq \beta + \frac{k-g}{g+1}$ approximation
- new complexity of $MA^r \approx (k2^{j_1} log(m))^g$ (Versus $(km)^g$)

1. Brief overview of classical *PTAS* design techniques
2. The guess approximation technique
3. Application to the *DRSSP*
   - Presentation of the problem
   - A first approximation scheme
   - Improved scheme with guess approximation
4. Discussion

Outline

Some questions about the guess approximation technique:

- does this technique lead to classical approximation schemes ?
- could we use the particular contraction function we introduced here for other problems ?
- are there some problems where this technique seems hard to apply ?
- can we get *FPTAS* for strongly *NP* complete problems

Outline

Some questions about the guess approximation technique:

- does this technique lead to classical approximation schemes ?
- could we use the particular contraction function we introduced here for other problems ?
- are there some problems where this technique seems hard to apply ?
- can we get *FPTAS* for strongly *NP* complete problems

Outline

Some questions about the guess approximation technique:

- does this technique lead to classical approximation schemes ?
- could we use the particular contraction function we introduced here for other problems ?
- are there some problems where this technique seems hard to apply ?
- can we get *FPTAS* for strongly *NP* complete problems

Outline

Some questions about the guess approximation technique:

- does this technique lead to classical approximation schemes ?
- could we use the particular contraction function we introduced here for other problems ?
- are there some problems where this technique seems hard to apply ?

- can we get *FPTAS* for strongly *NP* complete problems  ?

Thank you for your attention !!

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.

[4] H. Shachnai and T. Tamir.
Polynomial time approximation schemes - a survey.
Handbook of Approximation Algorithms and Metaheuristics, Chapman & Hall, Boca Raton, 2007.