Combining Multiple Heuristics on Discrete Resources

Marin Bougeret* , Pierre-François Dutot, Alfredo Goldman† , Yanik Ngoko‡ and Denis Trystram
LIG, Grenoble University, France

***Abstract*—In this work we study the portfolio problem which is to find a good combination of multiple heuristics to solve given instances on parallel resources in minimum time. The resources are assumed to be discrete, it is not possible to allocate a resource to more than one heuristic. Our goal is to minimize the average completion time of the set of instances, given a set of heuristics on homogeneous discrete resources. This problem has been studied in the continuous case in [13]. We first show that the problem is hard and that there is no constant ratio polynomial approximation unless $P = NP$ in the general case. Then, we design several approximation schemes for a restricted version of the problem where each heuristic must be used at least once. These results are obtained by using oracle with several guesses, leading to various tradeoff between the size of required information and the approximation ratio. Some additional results based on simulations are finally reported using a benchmark of instances on SAT solvers.**

I. Introduction

A. Description of the context and motivation

We are interested in this work in solving hard computational problems like the satisfiability problem SAT. It is well-established that a single algorithm cannot solve efficiently all the instances of such problems. In most cases, they are characterized by the great variability of their execution time depending on the considered instances. Thus, a good effective solution is to consider several heuristics and combine them in such a way to improve the mean execution time when solving a large set of instances. We are interested in this paper in designing adequate combination schemes. The suggested solution is based on the portfolio problem, introduced in the field of finance many years ago [11]. This problem can be informally recalled as follows: given a set of opportunities, an amount of possible investments on the set of opportunities and the payoff obtained when investing an amount on each opportunity, what is the best amount of investment to make on each opportunity in order to maximize the sum of the payoffs? We consider the simultaneous use of heuristics of a portfolio of instances on parallel resources. Using the vocabulary of Computer Science, we assume that there exists a benchmark composed of a finite set of instances and some heuristics which solve these instances. The expected execution times of heuristics on all the instances is known. The objective is to determine the best resource allocation for the heuristics such as to minimize the mean execution time of the set of instances. The execution time of an instance given a resource allocation is taken here as the shortest execution time of a heuristic when executing simultaneously all the heuristics on this instance following the resource allocation.

This formulation of the problem as a portfolio is motivated by the fact that we may not know which is the best suited heuristic to solve an instance before actually solving it. The interest of this sharing model is that in practice if the benchmark of instances is representative over all possible instances, we will have a better mean execution time than using only one heuristic.

B. Related works

There exist many studies focusing on the construction of automated heuristic selection process in various area. For a given problem, they usually proceed first by identifying the set of features which characterize its instances. A matching is then built between types of instances and heuristics in order to determine an efficient heuristic for any instance.

In [12] for example, the authors introduce a generic framework for heuristic selection in a parallel context and apply it to some classical problems like sorting, remote method invocation or parallel reductions. In [17], the authors suggest a model for the heuristic selection in the case of the resolution of partial differential equations. The Self Adapting Large-scale Solver Architecture (SALSA) project uses statistical techniques for solver selection [8]. In [6] the authors study the construction of a selection process for linear system resolution. The construction of automated selection process requires the identification of a representative set of features. This can be very difficult depending on problems [8].

There are other alternative works based on heuristic portfolio that can be used in these cases. A portfolio of heuristics is a collection of different algorithms (or algorithms with different parameters) running in an interleaved scheme. There are many other works interested by heuristic portfolio. In [9], [10], the authors show the interest to use heuristic portfolio on randomized heuristics. Concurrent use of heuristics for solving an instance have also been suggested in [7], [15] with the concept of asynchronous team. In [14], the authors study how to interleave the execution of different heuristics in order to reduce the execution time of a set of instances. We have mainly been interested in the resource sharing approach, as introduced in [13].

*This work is supported by a PhD grant from DGA-CNRS.
†This work was done on sabbatical leave from São Paulo University.
‡This work is supported by a grant from EGIDE, SCAC Yaounde-Cameroun.

The oracle based approach used in part V is inspired from many works. Indeed, the concept of adding some quantifiable information to study what improvements can be derived exists in other fields than optimization problems. Let us briefly describe them.

In the distributed context [5], a problem is called *information sensitive* if a few bits of information enable to decrease drastically the execution time. The information sensitiveness is used to classify problems by focusing on lower bounds on the size of advice necessary to reach a fixed performance, or giving explicitly oracle information and studying the improvement (as in [5] and [4]).

In the on-line context, this quantifiable oracle information could be recognized in the notion of "look-ahead". The look-ahead could be defined as a slice of the future which is revealed to the on-line algorithm. Thus, it is possible to prove lower and upper bounds depending on the size of this slice [2], [3].

In the context of optimization, some polynomial time approximation schemes have been designed thanks to the guessing technique [1]. This technique can be decomposed in two steps: proving an approximation ratio while assuming that a little part of the optimal solution is known, and finally enumerating the possibilities for this part of the optimal solution.

C. Contributions

In this paper we introduce a new problem (Discrete Resource Sharing Scheduling Problem, dRSSP) which is an extension of the continuous version presented in [13] (also based on heuristic portfolio [9], [10]). Our approach is motivated by the fact that an optimal non integer allocation as considered in [13] may not be feasible on an integer number of resources. We provide complexity and inapproximability results for this problem. Then, we study a restricted version of the dRSSP using an oracle based approach. We do not only apply the guessing technique, but we aim at developing a methodological approach by looking for several guesses and studying the different tradeoffs obtained. Finally, we run simulations (using instances extracted from actual execution time of heuristics solving the SAT problem) to evaluate these tradeoffs.

D. Organization of the paper

The rest of the paper is organized as follows. In Section II, we present the resource sharing problem as defined in [13]. In Section III, we introduce a discrete version of the resource sharing problem and we also study its complexity. In Section IV we prove the inapproximability of the problem, and define consequently a restricted version. We design in Section V several approximation schemes (based on oracle guesses) for this restricted problem. Finally, some experimental results of these approximation schemes are presented in Section VI. We conclude in Section VII.

II. Resource Sharing Scheduling Problem

A. Definition of the problem

We describe below the Resource Sharing Scheduling Problem introduced in [13]. As we have already indicated, the problem is characterized by a set of exhaustive, or representative, instances. Given many heuristics that can be used to solve these instances, a fraction of the whole resources has to be allocated to each of these heuristics in order to minimize the execution time of the set of instances. This problem can formally be described as follows:

Resource Sharing Scheduling Problem (RSSP)

Instance: A finite set of instances $I = \{I_1, \ldots, I_n\}$, a finite set of heuristics $H = \{h_1, \ldots, h_k\}$, a cost matrix $C(h_i, I_j) \in R^+$ for each $I_j \in I$, $h_i \in H$, a real value $T \in R^+$.

Question: Is there a vector $S = (S_1, \ldots, S_k)$ with $S_i \in [0,1]$ and $\sum_{i=1}^{k} S_i \leq 1$ such that $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \left\{ \frac{C(h_i, I_j)}{S_i} \text{ s.t } S_i > 0 \right\} \leq T$?

In this problem, $C(h_i, I_j)$ indicates the execution cost of heuristic h_i on the instance I_j with all the available resources. In practice, this cost can be considered as the execution time. The vector $S = (S_1, \ldots, S_k)$ with $S_i \in [0,1]$ defines a possible share indicating that each heuristic $h_i, 1 \leq i \leq k$ will be executed with a proportion S_i of the entire resources. In this problem, if a proportion S_i of the resources is allocated to a heuristic h_i, then h_i solve an instance I_j with a cost equal to $\frac{C(h_i, I_j)}{S_i}$. The resource sharing problem has been proved to be NP-complete in [13].

B. A related problem

Sayag et al. [13] have also studied another related problem, namely the time switching problem. This problem considers a finite set of instances and assume a finite set of interruptible heuristics. To solve instances, the execution of the different heuristics are interleaved in a fixed pattern of time intervals. As in RSSP an execution ends as soon as one heuristic solves the current instance.

As previously, the goal in the task switching problem is to find a schedule which minimizes the mean execution time on the set of instances. This approach is interesting in a single resource problem and has also been studied by Streeter et al. [14].

In [13], it has been proved that to each resource sharing schedule corresponds a time switching with a lower execution time. Even if the time switching approach produces schedules with a better execution time, it assumes that heuristics are interruptible. However, all the interrupted states have to be stored leading to a prohibitive memory cost on multiple resources.

Notice also that in several cases, giving more resources to a heuristic does not have a positive impact on its execution. This is especially true when using hard to parallelize heuristics, as those involving a large number of irregular memory accesses and communications [16]. That is why we focus on the discrete version of RSSP.

III. Discrete Resource Sharing Scheduling Problem

A. Definition of the problem

We consider now a finite number of discrete resources. Moreover, we initially do not assume a linear execution cost. The solution cost of an instance should be explicitly given for each heuristic and given number of resources. Formally, the problem can be stated as follows:

discrete Resource Sharing Scheduling Problem (dRSSP)

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

Question: Is there a vector $S = (S_1, \ldots, S_k)$ with $S_i \in \{0, \ldots, m\}$ and $0 < \sum_{i=1}^{k} S_i \leq m$ such that $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{C(h_i, I_j, S_i) | S_i > 0\} \leq T$?

The idea in this problem is to find an efficient partition of resources to deploy the set of heuristics on the homogeneous resources. The cost function ($\min_{1 \leq i \leq k} \{C(h_i, I_j, S_i)\}$) introduced by Sayag et al. [13] and used here, considers that for each instance, all the different heuristics are executed with the defined share and then stop their execution when at least one heuristic finds a solution.

We study from now on the dRSSP with the **linear cost assumption**, which indicates that the execution cost is proportional to the number of resources used ($C(h_i, I_j, p) = \frac{C(h_i, I_j, m)m}{p}$). For the sake of simplicity, we will denote $C(h_i, I_j, m)$ by $C(h_i, I_j)$. Thus, the cost function C is entirely defined by the $C(h_i, I_j), i \in \{1, \ldots, k\}, j \in \{1, \ldots, n\}$.

To emphasize the difference between dRSSP and RSSP we use the same example presented in [13]. Suppose that we have 2 instances (I_1, I_2), 2 heuristics (h_1, h_2), 2 resources and the following execution cost matrix $C = \begin{pmatrix} 2 & 10 \\ 10 & 1 \end{pmatrix}$.

To deduce the optimal solution with RSSP, a fraction x of one heuristic has to be allocated such as to minimize: $\min\left(\frac{2}{x}, \frac{10}{1-x}\right) + \min\left(\frac{10}{x}, \frac{1}{1-x}\right)$. The minimum is obtained for $x = 2 - \sqrt{2}$. Thus, the optimal solution with RSSP consists to give $2(2-\sqrt{2})$ resources to h_1 and $2(\sqrt{2}-1)$ resources to h_2. This solution leads to a schedule with a total execution time equal to 5.8284. The optimal solution with dRSSP consists to give one resource to h_1 and one to h_2. This solution gives a schedule with the total execution time equal to 6.

B. Complexity

Theorem 3.1: The dRSSP with linear cost assumption is NP-Complete

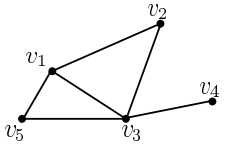
Proof: The problem clearly belongs to NP. We now build a reduction from the (well known) vertex cover problem to prove that the dRSSP problem is NP-complete.

Being given a graph $G = (V, E)$, $V = \{v_1, \ldots, v_k\}, k = |V|$, $|E| = n$ in which we are looking for a vertex cover $V^c \subseteq V$ of size m, we construct a dRSSP problem instance where $I = \{I_1, \ldots, I_n\}$, such that to each $I_j \in I$ corresponds an edge $(v_{j_1}, v_{j_2}) \in E$, $H = \{h_1, \ldots, h_k\}$ where h_i corresponds to a vertex,

$$C(h_i, I_j) = \begin{cases} \alpha > 0 & \text{if } v_i = v_{j_1} \text{ or } v_i = v_{j_2} \\ \beta & \text{where } \beta = nm\alpha + 1 \text{ otherwise.} \end{cases},$$

and $T = nm\alpha$.

Figure 1 presents an example of this reduction for a particular graph. Let us consider an instance of the Vertex Cover problem for which the corresponding dRSSP problem has a solution S.



Graph

I_1 — (v_1, v_2) I_4 — (v_3, v_4)
I_2 — (v_1, v_3) I_5 — (v_1, v_5)
I_3 — (v_2, v_3) I_6 — (v_5, v_3)

C	I_1	I_2	I_3	I_4	I_5	I_6
h_1	$\frac{1}{m}$	$\frac{1}{m}$	7	7	$\frac{1}{m}$	7
h_2	$\frac{1}{m}$	7	$\frac{1}{m}$	7	7	7
h_3	7	$\frac{1}{m}$	$\frac{1}{m}$	$\frac{1}{m}$	7	$\frac{1}{m}$
h_4	7	7	7	$\frac{1}{m}$	7	7
h_5	7	7	7	7	$\frac{1}{m}$	$\frac{1}{m}$

Execution cost matrix

Fig. 1. Example of derivation for a particular graph in which we are looking for a vertex cover of size m. In this example, $\alpha = \frac{1}{m}$

The set $V^c = \{v_i \text{ s.t } S_i \neq 0\}$ is of size at most m since we have m resources. This set is also a vertex cover since if there was $(v_{j_1}, v_{j_2}) \in E$ with $v_{j_1} \notin V^c$, $v_{j_2} \notin V^c$ then the cost of the associated problem would have been at least $nm\alpha + 1$.

On the other hand, if there exists a vertex cover V^c of size at most m then the vector $S = (S_1, \ldots, S_k)$ where $\forall i, 1 \leq i \leq k$ $S_i = \begin{cases} 1 & \text{if } v_i \in V^c \\ 0 & \text{otherwise,} \end{cases}$ is a solution to the above dRSSP problem. Each instance I_j can be treated using a heuristic corresponding to the considered vertex cover, and executed with one resource, leads to total cost $m\alpha$. Thus the cost of the dRSSP instance is at most equal to $nm\alpha$. ■

In the previous reduction, the corresponding dRSSP always has more heuristics than resources. An interesting question is to determine whether if dRSSP problem is still NP complete when there are more resources than heuristics.

Proposition 3.2: The dRSSP problem remains NP complete even when there are more resources than heuristics.

Proof: We will adapt the previous reduction in this case. The idea here is to introduce a virtual instance and a virtual heuristic such that we can reach the target bound only if we solve the virtual instance with a large number of resources with the virtual heuristic.

Being given a graph $G = (V, E)$, $V = \{v_1, \ldots, v_k\}, k = |V|$, $|E| = n$ in which we are looking for a vertex cover $V^c \subseteq V$ of size m, we construct a dRSSP instance with $(k+1)m$ resources, $I = \{I_1, \ldots, I_n, I_{n+1}\}$, such that to each $I_j \in I, j \leq n$ corresponds an edge (v_{j_1}, v_{j_2}) with $(v_{j_1}, v_{j_2}) \in E$, I_{n+1} is a virtual instance added, $H = \{h_1, \ldots, h_k, h_{k+1}\}$ where the $h_i, i \leq k$ corresponds to the vertices and h_{k+1} is a

virtual heuristic,

$$C(h_i, I_j) = \begin{cases} \alpha > 0 & \text{if } (i \leq k, j \leq n \text{ and} \\ & (v_i = v_{j_1} \text{ or } v_i = v_{j_2})) \\ k\gamma & \text{if } (i = k+1 \text{ and } j = n+1) \\ T+1 & \text{otherwise.} \end{cases}$$

Here $T = \gamma(k+1)+nm(k+1)\alpha$ and $\gamma = nm\alpha(km-1)+1$. We solve this problem by assuming that $(k+1)m$ resources are available and if there is a vector S satisfying the requirements of dRSSP, we take the set $V^c = \{v_i \text{ s.t } S_i \neq 0\}$ as a solution for the vertex cover problem. Otherwise, we answer that the given vertex cover problem does not have a solution. Notice that with this reduction, the number of heuristics $(k+1)$ is smaller than the number of resources $(k+1)m$. The γ value has been taken such that if there are less than km resources (for example $km-1$) which where assigned to h_{k+1}, then it will not be possible to have a portfolio of cost lower than T since the execution cost of I_{n+1} on h_{k+1} would be larger than T. Then it leads that any solution to this problem gives a solution to dRSSP on instances $I_1, \ldots, I_n$ with heuristics $h_1, \ldots, h_k$ on at most m resources and this corresponds to a solution for the vertex cover problem. ■

IV. Approximability

In this section, we will show that the dRSSP cannot be approximated within a constant factor. Therefore, we define a restricted version of the dRSSP for which we provide a first approximation algorithm.

Proposition 4.1: The dRSSP problem cannot be approximated within a constant factor in polynomial time, unless P=NP.

Proof: The gap reduction is directly deduced from the NP completeness proof. The only difference is that we choose $\beta = x\alpha nm$, for x in $\mathbb{R}^+$ (and $x > 1$). Then if there is a vertex cover of size m, the optimal cost of the corresponding dRSSP is $a_1 \leq nm\alpha$, otherwise the optimal cost is $a_2 \geq \beta$, and $\frac{a_2}{a_1} = x$, which can be arbitrarily large. ■

Since the case where the number of allocated resource can be zero has no polynomial approximation algorithm within a constant factor (unless $P = NP$), we consider from now on a restriction of the linear version of dRSSP in which each heuristic must use at least one processor, which requires $m \geq k$. This additional "allocating constraint" can be interpreted as a justification of the chosen portfolio of heuristics. Indeed, if in a solution we don't allocate any processor to a heuristic, it means that this heuristic shouldn't appear in the given portfolio. In other words, this constraint means that the portfolio is well chosen. We could also notice that if a heuristic is completely dominated by another (which means that for all the instances, the first one is slower than the second one), it would have no sense to allocate even one processor to the dominated heuristic. Thus, no heuristic is dominated by another in the given portfolio.

To solve this restricted version of the linear version of dRSSP, let us now analyze a greedy algorithm which will serve as a basis for more sophisticated approximations presented in the next section. We consider the algorithm **mean-allocation (MA)**, which consists in allocating $\lfloor \frac{m}{k} \rfloor$ processors to each heuristic.

Let us now define some new notations, given a fixed valid solution S (not necessarily produced by MA), and a fixed optimal solution S^*.

Definition 4.2: Let $\sigma(j) = argmin_{1 \leq i \leq k} C(h_i, I_j)/S_i$ be the index of the heuristic which finds the solution first for the instance j in S (ties are broken arbitrarily).
We define in the same way $\sigma^*(j)$ as the index of the used heuristic for the instance j in S^*.

Definition 4.3: Let $T(I_j) = \frac{C(h_{\sigma(j)}, I_j)}{S_{\sigma(j)}}$ be the processing time of instance j in S.
We define in the same way $T^*(I_j)$ as the processing time of instance j in S^*.

Proposition 4.4: **MA** is a k approximation for the restricted dRRSP.

Proof: Let us first remark that **MA** allocates at least one processor to every heuristic, which respects our new constraint. Let $(a, b) \in \mathbb{N}^2$ such that $m = ak + b, b < k$. Notice that $a \geq 1$, given that we must allocate at least one processor to every heuristic.

For any instance $j \in \{1, .., n\}$, we have $T(I_j) \leq \frac{C(h_{\sigma*(j)}, I_j)}{S_{\sigma^*(j)}}$ by definition of $T(I_j)$, and $\frac{C(h_{\sigma*(j)}, I_j)}{S_{\sigma^*(j)}} = \frac{S^*_{\sigma^*(j)}}{S_{\sigma^*(j)}} T^*(I_j) \leq \frac{m-(k-1)}{S_{\sigma^*(j)}} T^*(I_j)$ because in the worst case the considered optimal solution allocates the maximum possible number of resources to heuristic $\sigma^*(j)$. Finally, $\frac{m-(k-1)}{S_{\sigma^*(j)}} T^*(I_j) = \frac{ak+b-(k-1)}{a} T^*(I_j) \leq kT^*(I_j)$, which leads to the proposition.

■

We will now study how this algorithm can be improved thanks to the use of an oracle.

V. Oracle based approach

A. Introduction

In this section, we study the restricted version of dRSSP in a non-standard perspective by assuming the existence of a reliable oracle that can provide some extra information for each instance. We will not only apply the guessing technique, but have a methodological approach by looking for many possible guesses, and studying the different tradeoffs obtained. We show that by choosing "correctly" the asked information, it is possible to derive very good approximation ratio while simply using the MA algorithm. More specifically (for any $g \in \{1, \ldots, k-1\}$ and an execution time in $O(kn)$), we provide a $(k-g)$ approximation with an information of size [1] $glog(m)$, and a $\frac{k}{g+1}$ approximation with an information of size $g(log(k) + log(m))$. This kind of results leads to two interesting developments. First, it gives more insight into the considered problem because the type of information used emphasizes where the difficulty comes from. Secondly, this

[1] As the encoding of the instance is fixed, all the information sizes are given exactly, without using the O notation.

kind of approximation with advice is a first step towards the design of classical approximation algorithms by replacing the oracle with another algorithm (which for example enumerates all the possibilities). In both cases, this methodology reduces the original problem to the study of this oracle information.

B. Choosing an arbitrary subset of heuristics

As a first step, we choose arbitrarily g heuristics (denoted by $\{h_1, \ldots, h_g\}$ without loss of generality and called "the guessed heuristics") among the k available heuristics. In the first guess G_1, the oracle provides a part of an optimal solution: the oracle gives the number of processors allocated to these g heuristics in an optimal solution of the restricted $dRSSP$.

Definition 5.1 (Guess 1): Let $G_1 = (S_1^*, \ldots, S_g^*)$, for a fixed subset of g heuristics and a fixed optimal solution S^*.

Notice that this guess can be encoded using $|G_1| = glog(m)$ bits. We will study two algorithms, both based on MA, which make use of G_1. We first introduce some notations: let $k' = k - g$ be the number of remaining heuristics, $s = \Sigma_{i=1}^{g} S_i^*$ the number of processors used in the guess, and $m' = m - s$ the number of remaining processors. We also define $(a', b') \in \mathbb{N}^2$ such that $m' = a'k' + b', b' < k'$.

Let us consider a first algorithm MA^G which, given any guess $G = (X_1, \ldots, X_g), X_i \geq 1$, allocates X_i processors to heuristic $h_i, i \in \{1, \ldots, g\}$, and applies MA on the k' others heuristics with the m' remaining processors. This algorithm used with $G = G_1$ leads to the following ratio.

Proposition 5.2: MA^{G_1} is a $(k-g)$ approximation for the restricted dRSSP, for any $g \in \{0 \ldots k-1\}$.

Proof: First, remark that MA^{G_1} produces a valid solution because we know that $a' \geq 1$ (there is at least one processor per heuristic in the optimal solution considered). Then, for any instance j treated by a guessed heuristic in the optimal solution considered ($\sigma^*(j) \in \{1, \ldots, g\}$), MA^{G_1} is at least as good as the optimal. For the other instances, the analysis is the same as for the algorithm MA, and leads to the desired ratio. ∎

In the previous analysis, the approximation ratio for the instances treated by the guessed heuristics is unnecessarily good. Therefore, we will consider another algorithm, **mean-allocation-reassign** (MA_R^G), which redistribute a fraction of the processors allocated on guessed heuristics to the others. The goal will be of course to balance the tradeoff between the ratio for the guessed heuristics and the ratio for the others. A reasonable requirement for this algorithm to work is to consider that the number of processors allocated to the guessed heuristic is sufficiently large. Thereby, we need $s > k + c$ (with $c \geq 1$) to apply MA_R^G. We will discuss after the proof the implication of this assumption on the problem.

Let us define now more precisely the algorithm: given any guess $G = (X_1, \ldots, X_g), X_i \geq 1$, MA_R^G allocates $X_i - \lfloor \frac{X_i}{\alpha} \rfloor$ processors to heuristic $h_i, i \in \{1, \ldots, g\}$, and apply MA on the k' others heuristics with the $m' + \Sigma_{i=1}^{g} \lfloor \frac{X_i}{\alpha} \rfloor$ remaining processors. This algorithm used with $G = G_1$ leads to the following ratio, which is naturally decreasing according to the number s of processors allocated in the guess.

Proposition 5.3: For any guess G_1 such that $s > k + c$, there exists an α^* such that $MA_R^{G_1}$ is a

$$max(\frac{k+c}{c}, (k-g)(1 - \frac{\frac{k-g-1}{k-g}s - g - 1}{m-k}))$$

approximation for the restricted dRSSP.

Proof:

Let $\alpha \in \mathbb{R}^+$. We need $\alpha > 1$ to allocate at least one processor to the guessed heuristics. The ratio of any instance j such that $\sigma^*(j) \in \{1, \ldots, g\}$ is $\frac{\alpha}{\alpha - 1}$. Now we will bound the approximation ratio for the others heuristics. Let $x_0 = \lfloor \frac{b' + \Sigma_{i=1}^{g} \lfloor \frac{S_i^*}{\alpha} \rfloor}{k'} \rfloor$ be the number of processors added to each non guessed heuristic $h_j, j \in \{g+1, \ldots, k\}$. We have $x_0 \geq \frac{b' + \Sigma_{i=1}^{g} \lfloor \frac{S_i^*}{\alpha} \rfloor}{k'} - 1 \geq \frac{b' + \Sigma_{i=1}^{g} (\frac{S_i^*}{\alpha} - 1)}{k'} - 1 = \frac{\alpha b' + s - \alpha k}{\alpha k'}$. So we add at least $x = \frac{\alpha(b'-k)+s}{\alpha k'}$ processors per heuristic. The ratio for any instance j such that $\sigma^*(j) \in \{g+1, \ldots, k\}$ is bounded by $\frac{m' - (k'-1)}{a' + x}$. We have to ensure that $\alpha \leq \frac{s}{k - b'}$ to have $x \geq 0$. The two constraints are not conflicting because, according to the hypothesis, we know that $\frac{s}{k-b'} \geq \frac{s}{k} > 1$.

To summarize, we are looking for α^* which minimizes $max(\frac{\alpha}{\alpha - 1}, \frac{m' - (k'-1)}{a' + x})$.

$$\begin{aligned} \frac{\alpha^*}{\alpha^* - 1} &= \frac{m' - (k'-1)}{a' + \frac{\alpha^*(b'-k)+s}{\alpha^* k'}} \\ &= \frac{\alpha^* k'(m' - (k'-1))}{\alpha^*(m' - b') + \alpha^*(b' - k) + s} \\ \Longrightarrow \alpha^* &= \frac{k'(m' - (k'-1)) + s}{k'(m' - (k'-1)) + k - m'} \end{aligned}$$

We notice that $\alpha^* > 1$ is true as $m \geq s > k$. If $\alpha^* \leq \frac{s}{k-b'}$, we choose $\alpha = \alpha^*$, and we get the desired ratio $\frac{\alpha}{\alpha - 1} = \frac{k'(m' - (k'-1)) + s}{m - k} = (k-g)\frac{(m - (\frac{k-g-1}{k-g}s + k - g - 1))}{m-k}$.

In the other case ($\alpha^* > \frac{s}{k-b}$), we choose $\alpha = \frac{s}{k-b}$ (which also insure $\alpha > 1$), and the ratio is $\frac{\alpha}{\alpha - 1} \leq \frac{s}{s-k} \leq \frac{k+c}{c}$. ∎

Let us conclude this proof with a remark. To apply $MA_R^{G_1}$, remind that we need $s > k + c$ (c can be chosen in $\mathbb{N}^*$). To use this algorithm in a practical way, we would like to choose an arbitrary subset of g heuristics, and to try all the possible allocations for these heuristics. However, we don't know if the chosen heuristics satisfy $s > k + c$ in the optimal fixed solution S^*. Even worse, there could be instances where no such subset exists in S^* if g is not large enough (for example if $S_i^* = \frac{m}{k}, \forall i \in \{1, .., k\}$). Under certain assumptions ($m > k(k+c)$), one solution could be to modify the guess G_1 as follows: $G_1' = (S_1^*, ..., S_g^*)$, for an arbitrary subset of $g-1$ heuristics, and $S_1^* \geq S_i^*, \forall i \in \{2, .., k\}$. Thus, as we know that $s \geq S_1^* > \frac{m}{k}$, we could assert that $s > k + c$.

In the last solution, (where the only extra assumption we finally need is $m > k(k+c)$), we notice that we need particular properties for the chosen heuristics ($S_1^* \geq S_i^*$). Even if asking such properties increases the length of the guess

($|G'_1| = log(k) + glog(m)$ bits), it may lead to better approximation ratios. Thus, in the next part, instead of choosing an arbitrary subset of g (or $g-1$) heuristics, we will look for what could be the "best" properties to ask for.

C. Choosing a convenient subset of heuristics

In this part we come back to the restricted dRSSP (which is the dRSSP where the only extra constraint is to allocate at least one processor to each heuristic), and we define a new guess, which is larger than G_1, but leads to a better approximation ratio. Let us start with another analysis of the MA algorithm which underlines an interesting property. 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^*. We bound the value returned by MA by grouping the instances according to the heuristic on which they are computed in S^*:

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

The approximation ratio of the MA algorithm is $(2k-1)\frac{Max_i(T^*(h_i))}{Opt}$. >From this form of the ratio, we can infer that the difficulty in this problem comes from the input where Opt is close to $Max_i(T^*(h_i))$, *i.e.* when a very small number of heuristics is responsible for the major part of the total computing. Hence, we define the second guess as follows.

Definition 5.4 (Guess 2): Let $G_2 = (S_1^*, ..., S_g^*)$, be the number of processors allocated to the g most efficient heuristics (which means $T^*(h_1) \geq .. \geq T^*(h_g) \geq T^*(h_i), \forall i \in \{g+1,..,k\}$) in a fixed optimal solution S^*.

Notice that this guess can be encoded using $|G_2| = g(log(k)+log(m))$ bits to indicate which subset of g heuristics must be chosen, and the allocation of the heuristics.Thanks to this larger guess, we derive the following better ratio.

Proposition 5.5: MA^{G_2} is a $\frac{k}{g+1}$ approximation for the restricted dRSSP.

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

$$\begin{aligned} T_{algo} &= \sum_{i=1}^{g} \sum_{j/\sigma^*(j)=i} T(I_j) + \sum_{i=g+1}^{k} \sum_{j/\sigma^*(j)=i} T(I_j) \\ &\leq \sum_{i=1}^{g} \sum_{j/\sigma^*(j)=i} T^*(I_j) + \sum_{i=g+1}^{k} \sum_{j/\sigma^*(j)=i} \frac{S_i^*}{S_i} T^*(I_j) \\ &= \sum_{i=1}^{g} T^*(h_i) + \sum_{i=g+1}^{k} \frac{S_i^*}{S_i} T^*(h_i) \\ &= \sum_{i=1}^{k} T^*(h_i) + \sum_{i=g+1}^{k} (\frac{S_i^*}{S_i} - 1) T^*(h_i) \end{aligned}$$

Let us define $M = max_{i \in \{g+1,...,k\}}(T^*(h_i))$. Using the same notations ($m' = a'k' + b', a' \geq 1, b' < k'$), we get $T_{algo} \leq Opt + M(\frac{m'}{a'} - k') = Opt + M\frac{b'}{a'}$. Moreover, $Opt = \Sigma_{i=1}^{g} T^*(h_i) + \Sigma_{i=g+1}^{k} T^*(h_i) \geq gT^*(h_g) + M \geq (g+1)M$. Finally, the ratio for MA^{G_2} is $r \leq 1 + \frac{1}{g+1}\frac{b'}{a'} \leq 1 + \frac{k'-1}{g+1} = \frac{k}{g+1}$.

■

D. Summary

In this section, we investigated the restricted dRSSP using a methodology based on oracle guesses. We looked for what could be the more "efficient" guess to ask to the oracle, and obtained three particular tradeoff between the length of the guess and the derived ratio. These results give insight on what is difficult in this problem (finding which are the most used heuristics, and of course their allocation), and can be used to derive classical approximation schemes. The details of these schemes are indicated in figure 2.

algorithm	approximation ratio	complexity
MA^{G_1}	$(k-g)$	$O(m^g * kn)$
$MA_R^{G_1}$	$max \begin{cases} \frac{k+c}{c} \\ (k-g)(1-x) \end{cases}$	$O((k-g)m^g * kn)$
MA^{G_2}	$\frac{k}{g+1}$	$O((km)^g * kn)$

Fig. 2. Complexity of the oracle based approximation schemes, $MA_R^{G_1}$ requires $m > k(k+c)$ and $x = \frac{\frac{k-g-1}{k-g} s-g-1}{m-k}$

In the next section we present experimental results of our different approaches.

VI. Experimental results

We applied our algorithms on the satisfiability problem (SAT). The SAT problem consists in determining whether a formula of a logical proposition given as a conjunction of clauses is satisfied for at least one interpretation. Since this hard problem is very well known, there exist many heuristics that have been proposed to provide solutions for it.

For our experiments, we used a SAT database (SatEx[2]) which gives for a set of 23 heuristics (SAT solvers) and a benchmark of 1303 instances for SAT the CPU execution time (on a single machine) of each heuristics on the 1303 instances of the benchmark. Thus we didn't actually run these heuristics, but we used these CPU times to have a realistic matrix cost. Remember also that we have a linear cost assumption.

A. Benchmark

The 1303 instances of the SatEx database are issued from many domains where the SAT problem are encountered. Some of them are: Logistic planning, Formal verification of microprocessors, Scheduling. These instances are also issued

[2] http://www.lri.fr/~simon/satex/satex.php3

from many challenging benchmarks for SAT which are used in one of the most popular annual SAT competition [3].

B. Heuristics

The SatEx database contains 23 heuristics issued from three main SAT solvers family [18]. These are:

- The DLL family with the heuristics: $asat$, $csat$, $eqsatz$, $nsat$, $sat-grasp$, $posit$, $relsat$, $sato$, $sato-3.2.1$, $satz$, $satz-213$, $satz-215$, $zchaff$
- The DP family with : $calcres$, dr, $zres$,
- The randomized DLL family with : $ntab$, $ntab-back$, $ntab-back2$, $relsat-200$

Other heuristics are $heerhugo$, $modoc$, $modoc-2.0$

C. Simulations plan

To validate our algorithms, we did simulations with the set of heuristics and instances given by the SatEx database. In order to have several simulation cases (with cost matrix extracted from real execution time of heuristics solving the SAT problem), we both consider the entire set of heuristics and two randomly chosen subsets of heuristics. Given a heuristic h_i and an instance I_j of the database, let us denote by $cpu(h_i, I_j)$ the execution time of the heuristic h_i on the instance I_j. We did three series of simulations:

- In the first series of simulations (Experiments 1) we considered all the 23 heuristics of the SatEx database and we assumed that we have 100 resources. For each heuristic h_i, we took its execution time on 100 resources as equal to $cpu(h_i, I_j)$.
- In the second series (Experiments 2), we randomly selected a set of 9 heuristics among the complete set of 23 ones in the SatEx database and we assumed that we have 100 resources. For each heuristic h_i among the selected one, we took its execution time on 100 resources as equal to $cpu(h_i, I_j)$.
- In the third series (Experiments 3) , we randomly selected a set of 6 heuristics and we assumed that we have 50 resources. For each selected heuristic h_i , we took its execution time on 50 resources as equal to $cpu(h_i, I_j)$. The number of selected heuristics and the number of resources have been chosen such as to observe the behavior of our different approximation algorithms in comparison to the exact algorithm.

In all these experiments, we assumed the linear cost assumption.

D. Results

We present in Figure 3 the discrete resource sharing cost (sum of execution time over the set of instances) for MA, MA^{G_1} and MA^{G_2} in Experiments 1. Given a number g of heuristics to guess for MA^{G_1}, we did 20 simulations where we randomly selected (following a uniform distribution law) a subset of g heuristics between the 23 available. The values presented in Figure 3 for MA^{G_1} are the mean cost obtained from the 20 simulations and the standard deviation. In Figure 3 the MA^{G_2} algorithm gives the discrete resource sharing with the smallest cost for any value of g. One can notice that as suggested by our theoretical studies the discrete resource sharing cost of MA^{G_1} and MA^{G_2} decreases when the number of guessed heuristics is increased. One can also notice here that when guessing heuristics, the obtained resource sharing cost is better than those of the MA algorithm.

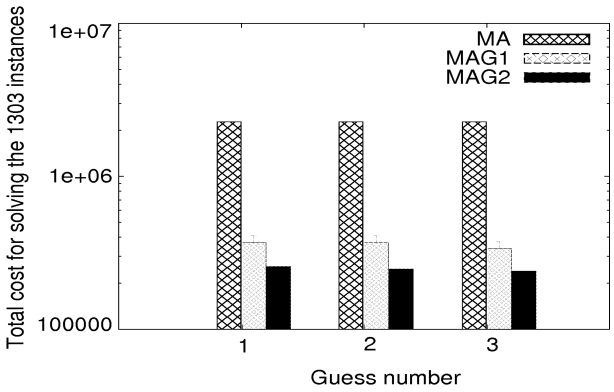


Fig. 3. Discrete Resource Sharing Cost with 23 heuristics and 100 resources

In Figure 4, we present the discrete resource sharing cost for, MA^{G_1}, $MA_R^{G_1}$, MA^{G_2} in Experiments 2. The subset of selected heuristics for this case is composed of: $satz$, $nsat$, $sato-3.2.1$, $satz-215$, $eqsatz$, $modoc-2.0$, $sato$, $modoc$, $posit$. Given a number g of heuristics to guess for MA^{G_1} and $MA_R^{G_1}$, we did 20 simulations where we randomly selected a subset of g heuristics between those considered. As in Experiments 1, one can notice here that the MA^{G_2} algorithm gives the discrete resource sharing with the smallest cost for any value of g. One can also notice here that the $MA_R^{G_1}$ returns a better value than the MA^{G_1} algorithm.

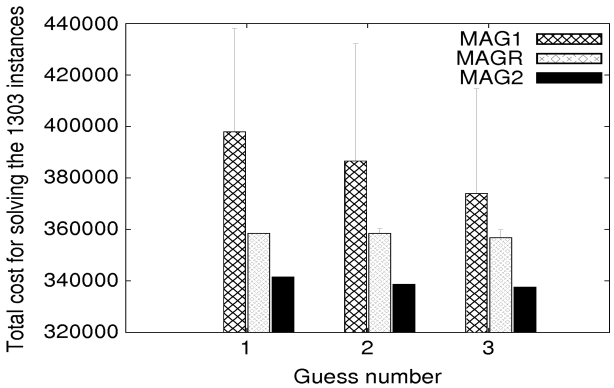


Fig. 4. Discrete Resource Sharing Cost with 9 heuristics and 100 resources

In Figure 5, we present the discrete resource sharing cost for, MA^{G_1}, $MA_R^{G_1}$, MA^{G_2} and the exact algorithm in Experiments 3. The subset of heuristics selected for this case is composed of: $csat$, $ntab-back2$, $modoc$, dr, $ntab$, $zchaff$. Given a number g of heuristics to guess for MA^{G_1} and

[3] http://www.satcompetition.org

$MA_R^{G_1}$, we did 20 simuations where we randomly selected a subset of g heuristics between those considered. In this Figure when guessing two heuristics, the MA^{G_2} algorithm provides a discrete resource sharing cost equal to those of the exact algorithm. This cost does not change for MA^{G_2} when guessing more heuristics. One can also notice here that the $MA_R^{G_1}$ returns a better value than the MA^{G_1} algorithm.

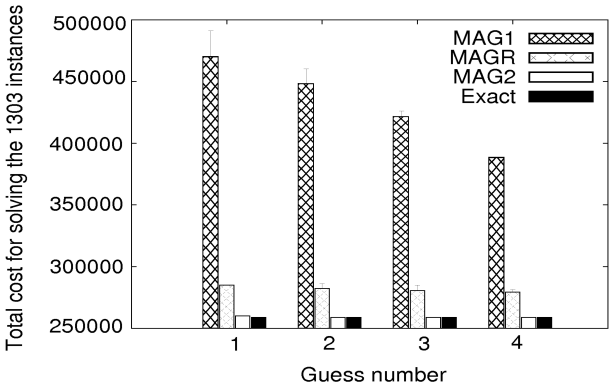


Fig. 5. Discrete Resource Sharing Cost with 6 heuristics and 50 resources

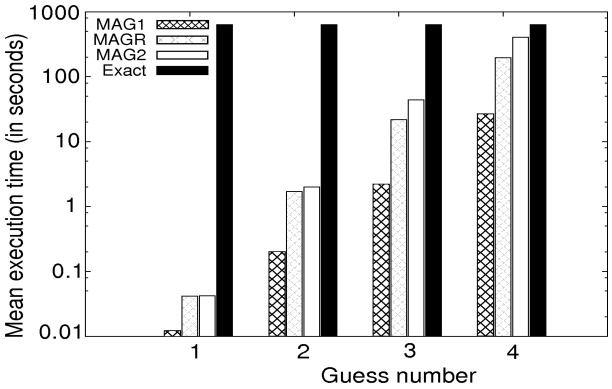


Fig. 6. Execution time with 6 heuristics and 50 resources

In Figure 6, we present the execution time necessary to compute our discrete resource sharing in Experiments 3. The exact solution is computed by brute force, MA^{G_1}, $MA_R^{G_1}$ and MA^{G_2} are executed with all the possible guesses of size g. Therefore all the execution times are exponential in g (the complexity of these algorithms is given in Figure 2, Section V).

These simulations were done on a AMD Opteron 246 cpu with 2 cores and 2 GB of memory. For MA^{G_1} and $MA_R^{G_1}$, we considered their mean execution times over the set of 20 simulations. This figure shows that the execution time grows when guessing more heuristics. We notice that for $g = 4$, the execution time of our approximation schemes is close to those of the exact algorithm. However remind that in this case we guess four of the six available heuristics (which is almost equivalent to the brute force algorithm), which is useless here since for $g = 2$, $MA_R^{G_1}$ is here an 1.1 approximation and MA^{G_2} provides the exact solution.

VII. Conclusion

In this document we introduced the discrete resource sharing scheduling problem. This problem, even restricted to linear costs, was shown to be NP-complete and does not have a polynomial time approximation algorithm within a constant factor, unless $P = NP$. However we provided approximation algorithms for the restricted version where each heuristic must be executed, based on oracles with well chosen guesses.

Following the theoretical analysis, some simulations have been conducted to study in detail the trade-off between the execution time and the size of the required information.

There are many perspectives to continue this work, namely the study of other cost functions; the proposal of new heuristics for the case of heterogeneous resources; and the study of a mixed problem with both resource sharing and time switching.

References

[1] H. Shachnai and T. Tamir – Polynomial Time Approximation Schemes - A Survey. – In Handbook of Approximation Algorithms and Metaheuristics (Ed. Teofilo F. Gonzalez), Chapman Hall/CRC Computer and Information Science Series, 2007.

[2] Susanne Albers – On the Influence of Lookahead in Competitive Paging Algorithms. – ALGORITHMICA, vol. 18, 283-305, 1997.

[3] Feifeng Zheng, Yinfeng Xu and E. Zhang – Oracle size : a new measure of difficulty for communication tasks. – Proceedings of the 25th ACM Symposium on Principles Of Distributed Computing (PODC),2006.

[4] Pierre Fraigniaud, David Ilcinkas and Andrzej Pelc – How much can lookahead help in online single machine scheduling. – Information Processing Letters (IPL), vol. 106, 2008.

[5] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas and Andrzej Pelc – Distributed Computing with Advice: Information Sensitivity of Graph Coloring. – Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP), 2007.

[6] S. Bhowmick et al. – Application of machine learning in selecting sparse linear. – To appear in: The International Journal of High Performance Computing Applications, 2006.

[7] V. Cicirello. – Boosting stochastic problem solvers through online self-analysis of performance. – PHD Thesis Carnegie Mellon University, 2003

[8] J. Dongarra et al. – Self-adapting numerical software (SANS) effort. – IBM Journal of Research and Development, vol. 50; 2-3, 223-238, 2006.

[9] C. Gomes and B. Selman. – Algorithm portfolios. – Artificial Intelligence Journal, 43-62, 2001.

[10] B. Huberman, R. Lukose and T. Hogg. – An economics approach to hard computational problems. – Science, 275: 51-54, 1997.

[11] Harry Markowitz. The early history of portfolio theory: 1600-1960, Financial Analysts Journal, 55 (4), 5-16, 1999.

[12] A. Ping et al. – STAPL: An adaptive, generic parallel C++ library. – International Workshop on Languages and Compilers for Parallel Computing, 193-208, 2001.

[13] T. Sayag, S. Fine and Y. Mansour. – Combining multiple heuristics. – Proceedings of the 23^{rd} International Symposium on Theoretical Aspects of Computer Science, 242-253, 2006.

[14] M. Streeter, D. Golovin and S. Smith. – Combining multiple heuristics online. – Proceedings of the 22^{rd} Conference on Artificial Intelligence, 1197-1203, 2007.

[15] S. Talukdar et al. – Asynchronous teams: Cooperation schemes for autonomous agents. – Journal of Heuristics, Vol. 4(1), 1998

[16] H. Yu and L. Rauchwerger. – Adaptive reduction parallelization. – Proceedings of the 14^{th} ACM conference on supercomputing, 66-77, 2000.

[17] S. Weerawarana et al. – PYTHIA: A knowledge-based system to select scientific algorithms. – ACM Transactions on Mathematical Software, vol. 22-4, 447-468, 1996.

[18] L. Simon. – SatEx: A knowledge-based system to select scientific algorithms. – ACM Transactions on Mathematical Software, vol. 22-4, 447-468, 1996.