International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Approximating the Discrete Resource Sharing Scheduling Problem

Marin Bougeret,[*] Pierre-François Dutot, Alfredo Goldman,[†] Yanik Ngoko[‡] and Denis Trystram

LIG, Grenoble University, 51 avenue J. Kuntzmann
38330 Montbonnot, France
FirstName.LastName@imag.fr

The goal of this work is to study the portfolio problem which consists in finding a good combination of multiple heuristics given a set of a problem instances to solve. We are interested in a parallel context where the resources are assumed to be discrete and homogeneous, and where it is not possible to allocate a given resource (processor) to more than one heuristic. The objective is to minimize the average completion time over the whole set of instances. We extend in this paper some existing analysis on the problem. More precisely, we provide a new complexity result for the restricted version of the problem, then, we generalize previous approximation schemes. In particular, they are improved using a guess approximation technique. Experimental results are also provided using a benchmark of instances on SAT solvers.

Keywords: resource allocation; combining algorithms; approximation schemes; oracle-guess approximation; SAT solver.

1. Introduction

1.1. *Description of the Portfolio problem*

We are interested in this work in solving hard computational problems like the satisfiability problem SAT [11]. It is well-established that a single algorithm cannot solve efficiently all the instances of such problems. In most cases, the algorithms 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. In this paper, we are interested in designing adequate combination schemes.

The suggested solution is based on the portfolio problem, introduced in the field of finance many years ago [14]. 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,

[*]Marin Bougeret has a PhD grant from DGA-CNRS
[†]Alfredo Goldman has been partially supported by the MINALOGIC project SCEPTRE funded by Region Rhone-Alpes in France
[‡]Yanik Ngoko was sponsored by the international exchange program EGIDE

what is the best amount of investment to make on each opportunity in order to maximize the sum of the payoffs? 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 possible resource allocation for the heuristics in order 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.

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 the set to be solved, we can expect a better mean execution time than using only one heuristic.

1.2. *Related works*

There exist many studies focusing on the construction of automated heuristic selection process. For a given problem, the approaches 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.

An et al. [2], for example, introduce a generic framework for heuristic selection in a parallel context and apply it to several classical problems including sorting and parallel reductions. Weerawarana et al. [21] suggest a model for the heuristics selection in the case of the resolution of partial differential equations. The SALSA project (Self Adapting Large-scale Solver Architecture) uses statistical techniques for solver selection [8]. Bhowmick et al. [3] study the construction of a selection process for solving linear systems. The construction of automated selection process requires the identification of a representative set of features. This can be very difficult depending on the targeted problems [8].

There exist 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. In [12, 13], the authors have demonstrated the interest to use heuristic portfolio on randomized heuristics. The concurrent use of heuristics for solving an instance has also been suggested in [7, 20] with the concept of asynchronous team. Streeter et al. [19] studied how to interleave the execution of various heuristics in order to reduce the execution time of a set of instances.

Sayag et al. [15] have also studied a related problem, namely the time switching problem. This problem considers a finite set of instances and assumes a finite set of interruptible heuristics. To solve instances, the execution of the various heuristics are interleaved in a fixed pattern of time intervals. The 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. [19]. Sayag et al. [15] proved that to each resource sharing schedule corresponds a time switching with a lower execution time, which means that if there is no overhead on context switching, it is always better to use time slicing on the heuristics on the whole resources instead of applying resource sharing. Even if the time switching approach produces theoretically schedules with constant approximation ratio, it assumes that the heuristics can be interrupted at any moment. However, all the interrupted states have to be stored, leading to a prohibitive memory cost on multiple resources.

Let us 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 heuristics that are hard to parallelize, like those involving a large number of irregular memory accesses and communications [22]. That is why we focus on the discrete version of the resource sharing scheduling problem (denoted by RSSP) instead of time switching as introduced in [15].

To the best of our knowledge, there are mainly two papers about the RSSP that are closely related to our work. Sayag et al.[15] address the continuous linear RSSP, where the output is a fraction of the available resources allocated to each heuristic, and the cost function is linear, meaning that the cost for solving an instance with only one resource is x times the cost with x resources. In our previous work [4] a discrete version of the linear restricted RSSP is studied, denoted by lr-dRSSP. The discrete setting means that the output is an integer representing the number of resources allocated to each heuristic, and the restriction assumption means that every solution must allocate at least one processor to every heuristic. The restricted version was proposed given that the l-dRSSP (without restriction) has no constant ratio polynomial approximation algorithm (unless $P = NP$) [4].

1.3. *Contributions and organization of the paper*

In this work, we study the r-dRSSP problem. More precisely, given any heuristic h_i, any instance I_j and any number of allocated resources in $\{1, \dots, m\}$, we only assume that $C(h_i, I_j, s)$ (the cost for solving I_j with h_i deployed on s resources) verifies $1 \leq \frac{C(h_i,I_j,s)}{C(h_i,I_j,\lambda s)} \leq \lambda, \forall \lambda \geq 1$. This reasonable assumption means that we handle any instance where the speedup is non "super linear".

A preliminary version for the lr-dRSSP problem was published in the APDCM workshop [4]. The r-dRSSP problem is defined in Section 2.1. We provide in Section 2.2 a proof for the complexity of lr-dRSSP (which was still open in [4]). In Section 2.3, a first greedy algorithm is provided to give an insight on the problem. Oracle-based algorithms are then introduced in Section 3.1. We extend in 3.2 and 3.3 the oracle based approximation schemes proposed in [4] for lr-dRSSP to r-dRSSP. We give in 3.4 a kind of tightness result by proving that the question asked to the oracle is almost the most "efficient" possible one. Then, we improve in Section 3.5 these approximation schemes by applying the guess approximation technique (introduced and applied to lr-dRSSP in [6]). Finally, in Section 4 we run experiments (using instances extracted from actual execution times of heuristics solving the SAT problem) to evaluate these algorithms.

2. Discrete Resource Sharing Scheduling Problem

2.1. *Definition of the problem*

restricted discrete Resource Sharing Scheduling Problem (r-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 $C(h_i, I_j, s) \in R^+$ for each $I_j \in I$, $h_i \in H$ and $s \in \{1, \ldots, m\}$, such that $1 \leq \frac{C(h_i,I_j,s)}{C(h_i,I_j,\lambda s)} \leq \lambda, \forall \lambda \geq 1$

Output: An allocation of the m resources $S = (S_1, \ldots, S_k)$ such that $S_i \in \mathbb{N}$ and $\sum_{i=1}^{k} S_i \leq m$ that minimizes $\sum_{j=1}^{n} \min_{1 \leq i \leq k} \{C(h_i, I_j, S_i) | S_i > 0\}$

The idea in this formulation 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. [15] 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.

2.2. *Complexity*

The NP completeness proof of the l-dRSSP is provided in [4], using a reduction from the vertex cover problem. However, this proof cannot be directly adapted to lr-dRSSP. Indeed, the gap between the optimal and non-optimal cases comes from the fact that it is impossible to give one resource to every "important" heuristic when there is no vertex cover, whereas the new constraint in lr-dRSSP forces any solution to allocate at least one resource to every heuristic. Thus, we add in the new reduction (still from the vertex cover) some artificial instances that amortize this new constraint.

Theorem 1. *lr-dRSSP is NP hard.*

Proof. First, let remark that it is straightforward to verify that the problem is in NP. Let us denote by T the threshold value of the lr-dRSSP decision problem. Given any solution S and any subset $X \subset I$, we denote by $f(S)_{|X} = \sum_{I_j \in X} \min_{1 \leq i \leq k} \{C(h_i, I_j, S_i) | S_i > 0\}$ the restricted cost of S on X. 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 x, we construct an instance of the lr-dRSSP problem as follows. We define the set of heuristics as $H = \{h_1, \ldots, h_k\}$ (to each h_i corresponds the vertex $v_i \in V$). The number of resources is $m = k + x$. Given that any solution of lr-dRSSP must allocate at least one resource to each heuristic, the decision only concerns the x extra resources. We choose $I = I' \cup I''$, with $I' = \{I_1, \dots, I_n\}$ (to each $I_j \in I'$ corresponds an edge $(v_{j_1}, v_{j_2}) \in E$), and $I'' = \{I_{n+1}, \dots, I_{n+k}\}$ (I'' forces the solutions to balance the x "extra" resources, which means not giving more than 2 resources to anybody). According to the linear cost assumption, it is sufficient to define only the $C(h_i, I_j, m)$ values. The cost function is:

$$C(h_i, I_j, m) = \begin{cases} \alpha & \text{if } j \in \{1, \dots, n\} \text{ and } v_i = v_{j_1} \text{ or } v_i = v_{j_2} \\ Z & \text{if } j \in \{n+1, \dots, n+k\} \text{ and } n+i = j \\ \beta & \text{otherwise} \end{cases}$$

The constant $\alpha > 0$ is chosen arbitrarily, and $\beta = mT + 1$ so that if a heuristic computes an instance with this cost (even with all the resources), the corresponding solution cannot be under the threshold value T. The values of T and Z later will be fixed later. The basic idea of this reduction is the same as for the l-dRSSP problem: if there is a vertex cover of size x, then there is a good naive solution S^1 which gives one "extra" resource to each of the x corresponding heuristics. Otherwise, we know that any potentially good solution S must balance the x extra resources (I'' forces any solution to well-balance the x), and then it is easy to see that S^1 is better than S on I'.

More precisely, if there is a vertex cover $V^c = \{v_{i_1}, \dots, v_{i_x}\}$ of size x, we define $S^1_i = 2$ if $v_i \in V^c$, and 1 otherwise. We have $\sum_{i=1}^{k} S^1_i = k + x$, and $Opt \leq f(S^1) = f(S^1)_{|I'} + f(S^1)_{|I''} = nm\frac{\alpha}{2} + Zm(k - x + \frac{x}{2})$. Thus, we define the threshold value $T = nm\frac{\alpha}{2} + Zm(k - x + \frac{x}{2})$. This value still depends on Z, which will be defined after.

Otherwise, let us consider a solution S, and let $a = card\{S_i = 1\} = k - x + j, j \in \{0, \dots, x-1\}$. Because of I'', a should not be too large. We proceed by cases according to the value of j.

If $j > 0$, $f(S) \geq f(S)_{|I''} = mZ(a + \Sigma_{S_i \neq 1} \frac{1}{S_i})$. Given that the function $t(x) = \frac{1}{x}$ is convex, we know that $\Sigma_{S_i \neq 1} t(S_i) \geq (x - j) t(\frac{\Sigma_{S_i \neq 1} S_i}{x - j}) = (x - j) t(\frac{2x - j}{x - j})$, which implies $f(S) \geq mZ(k - x + j + \frac{(x-j)^2}{2x - j})$. Hence,

$$f(S) - T \geq m(Z(j + \frac{(x-j)^2}{2x-j} - \frac{x}{2}) - \frac{n\alpha}{2})$$

$$\begin{aligned} f(S) - T &\geq m(Z(\frac{xj}{2(2x-j)}) - \frac{n\alpha}{2}) \\ &\geq m(\frac{Z}{4} - \frac{n\alpha}{2}) \text{ because } (j \geq 1) \end{aligned}$$

Thus, we define $Z = 2n\alpha + 1$, and we get $f(S) - T > 0$. So in this first case, the considered solution S is strictly over the threshold value T. In the other case ($j = 0$), $f(S) = f(S)_{|I'} + f(S)_{|I''} \geq (n-1)m\frac{\alpha}{2} + m\alpha + f(S^1)_{|I''}$. Then, $f(S) - T \geq \frac{\alpha}{2} > 0$. In both cases, if there is no vertex cover of size x, the cost of any solution S is strictly greater than T which implies $Opt > T$. □

Notice that in Theorem 1 we proved the NP hardness using a linear cost function, which means that for all i, j and for every number of allocated resources $s \in \{1, \ldots, m\}$, $C(h_i, I_j, s) = C(h_i, I_j, m)\frac{m}{s}$. This theorem implies of course that the r-dRSSP is also NP hard, through the "natural" reduction which consists in writing explicitly the m values for each heuristic and each instance. However, one could argue that it is sufficient in the linear case to only give the $C(h_i, I_j, m)$ values, implying that the input could be described using only $O(nk\log(Max_{(y_1,y_2)}(C(y1, y2, m))) + \log(m))$ bits. Then, the previous reduction from lr-dRSSP to r-DRSSP would not be polynomial in $\log(m)$, but only in m. However, this $O(m)$ dependencies is not important because Theorem 1 even proves that lr-dRSSP is unary NP hard.

2.3. *A first greedy algorithm: mean-allocation (MA)*

To solve r-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 introduce some new definitions and notations, given a fixed valid solution S (not necessarily produced by MA), and a fixed optimal solution S^*.

Definition 2. *Let $\sigma(j) = argmin_{1 \le i \le k} C(h_i, I_j, S_i)$ be the index of the heuristic which finds the solution first for the instance j in S (ties are broken arbitrarily). Let define in the same way $\sigma^*(j)$ as the index of the used heuristic for the instance j in S^*.*

Definition 3. *Let $T(I_j) = C(h_{\sigma(j)}, I_j, S_{\sigma(j)})$ be the processing time of instance j in S. Let define in the same way $T^*(I_j)$ as the processing time of instance j in S^*.*

Proposition 4. ***MA** is a k-approximation for r-dRSSP.*

Proof. Let a and b in $\mathbb{N}$ such that $m = ak + b, b < k$. Notice that $m \ge k$, otherwise there are no feasible solutions where each heuristic has at least one processor. Hence, a is greater or equal to 1.

For any instance $j \in \{1, .., n\}$, we have $T(I_j) = C(h_{\sigma(j)}, I_j, S_{\sigma(j)}) \le C(h_{\sigma^*(j)}, I_j, S_{\sigma^*(j)})$ by definition of $T(I_j)$. In the worst case, $S_{\sigma^*(j)} \le S^*_{\sigma^*(j)}$, implying $C(h_{\sigma^*(j)}, I_j, S_{\sigma^*(j)}) \le \frac{S^*_{\sigma^*(j)}}{S_{\sigma^*(j)}} C(h_{\sigma^*(j)}, I_j, S^*_{\sigma^*(j)}) \le \frac{m-(k-1)}{S_{\sigma^*(j)}} T^*(I_j)$ because in the worst case, the considered optimal solution allocates the maximum possible number of resources to heuristic $\sigma^*(j)$. Finally,

$$\frac{m - (k-1)}{S_{\sigma^*(j)}} T^*(I_j) = \frac{ak + b - (k-1)}{a} T^*(I_j) \le kT^*(I_j). \qquad \square$$

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

3. Approximation schemes based on oracle

3.1. *Introduction*

In this section, we provide approximation schemes for r-dRSSP using the oracle formalism: we first assume the existence of a reliable oracle that can provide some extra information for each instance, which allows to derive good approximation ratios, and finally we will enumerate the set of the possible oracle answers to get a "classical" algorithm (without oracle). The concept of adding some quantifiable information to study what improvements can be derived exists in other fields than optimization. Let us briefly review these oracle techniques.

In the distributed context [9], 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 (like in [9] and [10]).

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 [1, 23].

In the context of optimization, some polynomial time approximation schemes have been designed thanks to the guessing technique [17, 16]. 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 all the possibilities for this part of the optimal solution.

In this section, we show that by choosing "correctly" the asked information, it is possible to derive very good approximation ratio, even with MA as a basis. Moreover, we will not only apply the guessing technique, but have a methodological approach by proving that the question asked to the oracle is somehow "the best" possible. At last, we improve the obtained approximation scheme by using the guess approximation technique [6]. More precisely, we provide (for any $g \in \{1, \dots, k-1\}$ and an execution time in $O(kn)$) a $(k-g)$-approximation with an information of size[a] $g\log(m)$, and a $\frac{k}{g+1}$-approximation with an information of size $g(\log(k)+\log(m))$. Then, we prove that no information of "the same type" (coupled with MA) could lead to a better ratio than $\frac{k-g}{g+1}$. Finally, aiming at reducing the size of the oracle answer, we provide a ρ-approximation with an information of size $g(\log(k)+j_1+\log(\log(m)))$, where ρ is defined as:

$$\rho = \frac{k + \frac{g}{2^{j_1-1}}}{g+1}.$$

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

3.2. *Choosing an arbitrary subset of heuristics*

As a first step, we choose arbitrarily g heuristics (denoted by $\{h_{i_1}, \dots, h_{i_g}\}$ and called "the guessed heuristics") among the k available heuristics. In the first guess $\tilde{G}_1$, 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 r-dRSSP.

Definition 5 (Guess 1) *Let $\tilde{G}_1 = (S^*_{i_1}, \dots, S^*_{i_g})$, for a fixed subset $(i_1, \dots, i_g)$ of g heuristics and a fixed optimal solution S^*.*

Notice that this guess can be encoded using $|\tilde{G}_1| = g\log(m)$ bits. We first introduce some notations: let $k' = k - g$ be the number of remaining heuristics, $s = \Sigma^g_{l=1} S^*_{i_l}$ 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 $G = [(i_1, \dots, i_g)(X_1, \dots, X_g)], X_i \geq 1$, allocates X_l processors to heuristic $h_{i_l}, l \in \{1, \dots, g\}$, and applies MA on the k' other heuristics with the m' remaining processors. This algorithm used with $G_1 = [(i_1, \dots, i_g)(\tilde{G}_1)]$ leads to the following ratio.

Proposition 6. *MA^{G_1} is a $(k - g)$-approximation for r-dRSSP, for any $g \in \{0 \dots k - 1\}$.*

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 considered optimal solution). Then, for any instance j treated by a guessed heuristic in the considered optimal solution ($\sigma^*(j) \in \{1, \dots, g\}$), MA^{G_1} is at least as good as the optimal. For the other instances, the analysis is the same as for MA, and leads to the desired ratio. □

Corollary 7. *There is an $(k-g)$-approximation for r-dRSSP which runs in $O(m^g * kn)$, for any $g \in \{0 \dots k - 1\}$.*

Proof. We simply enumerate all the possible answers of the oracle, and choose the best solution, incurring a cost in $O(2^{|\tilde{G}_1|} * kn)$. □

In the following, instead of choosing an arbitrary subset of g heuristics, we will look for what could be the "best" properties to ask for.

3.3. *Choosing a convenient subset of heuristics*

In this section we define a new guess, which is larger than $\tilde{G}_1$, but leads to a better approximation ratio. As shown in [4], the "difficult" instances seem to be the ones where the optimal solution uses only a few heuristics (meaning that the major part of the computation time is only due to these few heuristics). For example, the worst cases of MA and MA^{G_1} occur when the optimal uses only one heuristic and allocates almost all the resources to it. Hence, we are interested in the most useful heuristics and we introduce the following definition. 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 define the second guess as follows.

Definition 8 (Guess 2) *Let $G_2 = [(i_1^*, \dots, i_g^*), (S^*_{i_1^*}, ..., S^*_{i_g^*})]$, be the number of processors allocated to the g most efficient heuristics (which means $T^*(h_{i_1^*}) \geq .. \geq T^*(h_{i_g^*})$ and $T^*(h_{i_g^*}) \geq T^*(h_i), \forall i \notin \{i_1^*, .., i_g^*\}$) 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 9. *MA^{G_2} is a $\frac{k}{g+1}$-approximation for r-dRSSP.*

Proof. For the sake of clarity, we can assume without loss of generality that the heuristics indicated by the oracle are the g first one, meaning that $(i_1^*, \dots, i_g^*) = (1, \dots, g)$. The proof with an arbitrary cost function is structured as the one in [4] with linear cost assumption. Let $X^+ = \{i \in \{g+1, \dots, k\} | S_i > S_i^*\}$ be the set of "non-guessed" heuristics for which MA^{G_2} allocated more than the optimal number of resources. We define in the same way $X^- = \{i \in \{g+1, \dots, k\} | S_i \leq S_i^*\}$.

For all the instances j such that $\sigma^*(j)$ is in $\{1, \dots, g\}$ or X^+, we have $T(I_j) \leq T^*(I_j)$ as MA^{G_2} allocates at least the number of resources used in the optimal for these instances. For j such that $\sigma^*(j)$ is in X^-, we have $T(I_j) \leq \frac{S_i^*}{S_i} T^*(I_j)$ since the cost function is not super-linear. Thus, summing over all heuristics we get:

$$\begin{aligned} T_{MA^{G_2}} &\leq \sum_{i=1}^{g} T^*(h_i) + \sum_{i \in X^+} T^*(h_i) + \sum_{i \in X^-} \frac{S_i^*}{S_i} T^*(h_i) \\ &= \sum_{i=1}^{k} T^*(h_i) + \sum_{i \in X^-} \left(\frac{S_i^*}{S_i} - 1 \right) T^*(h_i) \end{aligned}$$

$$T_{MA^{G_2}} \leq Opt + M \underbrace{\left(\frac{\sum_{i \in X^-} S_i^*}{a'} - card(X^-) \right)}_{\lambda}, \text{ with } M = \max_{i \in \{g+1, \dots, k\}} (T^*(h_i))$$

Then, we claim that $\lambda \leq k' - 1$. Let $j = card\{X^-\}$. We can assume $j \geq 1$, otherwise MA^{G_2} is optimal. We use the same notations ($m' = a'k' + b'$ with $a' \geq 1$, $b' < k'$) as in the previous proof. The worst case occurs when the optimal solution allocates only one resource to each heuristic in X^-, leading to $\sum_{i \in X^-} S_i^* = m' - \sum_{i \in X^+} S_i^* \leq m' - (k' - j)$. Thus, $\lambda \leq \frac{m' - k' + j(1-a')}{a'} \leq \frac{a'k' + b' - (k'-1) - a'}{a'} \leq k' - 1$.

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 $r \leq 1 + \frac{k'-1}{g+1} = \frac{k}{g+1}$. □

Corollary 10. *There is an $\frac{k}{g+1}$-approximation for r-dRSSP which runs in $O((km)^g * kn)$, for any $g \in \{0 \dots k-1\}$.*

3.4. *Tightness: comparison with the best subset of heuristics*

In the previous subsection, we investigated a "convenient" property to decide which subset of heuristics should be asked to the oracle. By asking the allocation of a particular subset of g heuristics as proposed in Definition 8 (rather than an arbitrary subset of size g), we improved the $(k-g)$-approximation ratio to a $\frac{k}{g+1}$ ratio. This drastic improvement leads to a natural question: is there another property that would lead to really better approximation ratio than $\frac{k}{g+1}$? As we will explain in Proposition 11, we investigated this question and found that no property leads to a better ratio than $\frac{k-g}{g+1}$.

Proposition 11 (Tightness) *There is no selection property (i.e. a criteria to select for which subset of g heuristics we ask the optimal allocation to the oracle) that leads to a better ratio than $\frac{k-g}{g+1}$, even for a linear cost function.*

Proof. To prove this statement, we construct an instance such that whatever the selected subset of g heuristics $(i_1,\dots,i_g)$, the corresponding guess $G = [(i_1,\dots,i_g)(S^*_{i_1},\dots,S^*_{i_g})]$ is such that $MA^G \geq \frac{k-g}{g+1}Opt$.

Let us define now this particular instance X, for fixed size of guess g (we use the same notation $k' = k-g$). We consider k heuristics, $m = (g+2)k'-1$ resources, and a set $I = I' \cup I''$ of k instances, with $I' = \{I_1,\dots,I_{g+1}\}$ and $I'' = \{I_{g+2},\dots,I_k\}$. Given that we are under the linear cost assumption, it is sufficient to only define the $C(h_i, I_j, m)$ values. The cost function is:

$$C(h_i, I_j, m) = \begin{cases} Z & \text{if } j \in \{1,\dots,g+1\} \text{ and } i=j \\ \epsilon & \text{if } j \in \{g+2,\dots,k\} \text{ and } i=j \\ \beta & \text{otherwise.} \end{cases}$$

We choose $\beta = mZ+1$ so that for any solution S the cost for solving I_j only depends on S_j (because $\forall S, \sigma(j)=j$).

We claim that the optimal solution S^* for X_c allocates k' resources to heuristic $h_i, i \in \{1,\dots,g+1\}$, and 1 resource $h_i, i \in \{g+2,\dots,k\}$. Notice that the total number of allocated resources is equal to m. The cost of S^* is $f(S^*) = m((g+1)\frac{Z}{k'} + (k'-1)\epsilon)$. We will now prove that this solution is optimal with a similar argument as in the NP hardness proof: any "good" solution must allocate as many resources as possible (*ie* $(g+1)k'$) to solve the first $g+1$ instances, and moreover these resources must be evenly distributed to the first $g+1$ heuristics. More precisely, consider a solution $S = \{S_1,\dots,S_k\}$ such that $\exists i_0 \in \{1,\dots,g+1\}/S_{i_0} = k'-j, j \geq 1$. Then, $f(S) \geq f(S)_{|I'} = m(\frac{Z}{S_{i_0}} + \Sigma_{i\in\{1,\dots,g+1\}, i\neq i_0}\frac{Z}{S_i})$. Using the same argument of convexity, we get $f(S) \geq mZ(\frac{1}{S_{i_0}} + g\frac{g}{gk'+j})$ Finally, $f(S) - f(S^*) \geq m(Z(\frac{1}{k'-j} + \frac{g^2}{gk'+j} - \frac{g+1}{k'}) - (k'-1)\epsilon) = m(Z\frac{j^2(g+1)}{k'(gk'+j)(k'-j)} - (k'-1)\epsilon) > 0$ for Z arbitrary large. Hence, S^* is an optimal solution for X.

Let us now analyze the cost of MA^G for any subset $\{i_1, \dots, i_g\}$. Let $x = Card(\{i_1, \dots, i_g\} \cap \{g+2, \dots, k\})$. Notice that $0 \le x \le min(g, k'-1)$ The total number of resources in the guess is $s = (g-x)k' + x$, and $m' = (x+1)k' + k' - 1 - x$, implying $m' < (x+2)k'$. Then, the cost of MA^G (under the linear cost assumption) is $T_{MA^G} \ge m((g-x)\frac{Z}{k'} + (x+1)\frac{Z}{\alpha})$, with $\alpha \le (x+1)$. Thus, $T_{MA^G} \ge m((g-x)\frac{Z}{k'} + Z) \ge mZ$, and $\frac{T_{MA^G}}{Opt} \ge \frac{Z}{\frac{(g+1)Z}{k'} + (k'-1)\epsilon} \xrightarrow[Z \to \infty]{} \frac{k'}{g+1}$. □

Notice that the previous instance could be used to prove that the $\frac{k}{g+1}$ ratio of MA^{G_2} is tight. Indeed, with Z large enough we will have $G_2 = [(1, 2, \dots, g-1, g), (k', \dots, k')]$ and $\frac{T_{MA^{G_2}}}{Opt} \ge \frac{g\frac{Z}{k'} + Z}{(g+1)\frac{Z}{k'} + (k'-1)\epsilon} \xrightarrow[Z \to \infty]{} \frac{k}{g+1}$.

3.5. *Improvements with guess approximation*

The idea of the guess approximation technique (introduced in [6]) is to find how to "contract" the answer of the oracle, without neither loosing too much information nor worsening the approximation ratio. We now look for what could be contracted here. In the case of guess 2, the oracle provides two types of information: a set of indices of heuristics (which verify a particular property) and a set of numbers of resources. The first type of information seems to be hard to approximate. Indeed, the information here is intrinsically binary, that is a given heuristic satisfies or not a given property. Thus, we are more interested in approximating the second part of the oracle information: the number of processors allocated to particular heuristics (in an optimal solution). Using the same notation as in Definition 8, let $\bar{G_2} = [(i^*_1, \dots, i^*_g), (\bar{S^*_{i^*_1}}, ..., \bar{S^*_{i^*_g}})]$ be the contracted answer.

In order to define "correctly" the $\bar{S^*_{i_l}}$, let us consider the two following points. First, notice that it could be wrong to allocate more processors than the optimal for the guessed heuristics (*i.e.* $\sum_{l=1}^{g} \bar{S^*_{i_l}} > \sum_{l=1}^{g} S^*_{i_l}$), because there could be not enough remaining resources to allocate to the "non-guessed" heuristics, leading to an infeasible solution. Thus, the contraction can be chosen such that $\sum_{l=1}^{g} \bar{S^*_{i_l}} \le \sum_{l=1}^{g} S^*_{i_l}$ (we will even choose $\bar{S^*_{i_l}} \le S^*_{i_l}, \forall l$). Secondly, using only x resources instead of $x + \epsilon$ could be very bad if x is small.

Taking into account these two remarks, let us define the $\bar{S^*_{i_l}}$ as follows. We consider a mantissa-exponent representation for the $S^*_{i_l}$, with a fixed size[b] $j_1 \in \{1, .., \lceil \log(m) \rceil\}$ of mantissa. Thus, $S^*_{i_l} = a_{i_l} 2^{e_{i_l}} + b_{i_l}$, where a_{i_l} is encoded on j_1 bits, $0 \le e_{i_l} \le \lceil \log(m) \rceil - j_1$, and $b_{i_l} \le 2^{e_{i_l}} - 1$. Then, we define $\bar{S^*_{i_l}} = a_{i_l} 2^{e_{i_l}}$. Notice that the length of the approximated guess becomes $|\bar{G_2}| = \Sigma_{l=1}^{g}(\log(i_l) + |a_{i_l}| + |e_{i_l}|) \le g(\log(k) + j_1 + \log(\log(m)))$. We now study the approximation ratio derived with the same algorithm as previously.

Proposition 12. *$MA^{\bar{G_2}}$ is a $\frac{k + \frac{g}{2^{j_1}-1}}{g+1}$-approximation for r-dRSSP.*

[b]In all the paper, log denote the $\log_2$ function

Proof. Again, let us assume without loss of generality that $(i^*_1, \dots, i^*_g) = (1, \dots, g)$. We first prove that $\forall i \in \{1, \dots, g\}, S^*_i / \bar{S^*_i} \leq \beta$, with $\beta = 1 + \frac{1}{2^{j_1 - 1}}$. If $S^*_i \leq 2^{j_1} - 1$, we have $\bar{S^*_i} = S^*_i$ because S^*_i can be written with j_1 bits. Otherwise, $S^*_i / \bar{S^*_i} = \frac{a_i 2^{e_i} + b_i}{a_i 2^{e_i}} \leq 1 + \frac{1}{a_i} \leq 1 + \frac{1}{2^{j_1 - 1}} = \beta$.

Then, the proof can be directly adapted from Proposition 9 (we use the same notation X^+ and X^-):

$$\begin{aligned}
T_{MA^{\bar{G}_2}} &\leq \beta \sum_{i=1}^{g} T^*(h_i) + \sum_{i \in X^+} T^*(h_i) + \sum_{i \in X^-} \frac{S^*_i}{S_i} T^*(h_i) \\
&= \beta \sum_{i=1}^{k} T^*(h_i) + \sum_{i \in X^+} (1 - \beta) T^*(h_i) + \sum_{i \in X^-} (\frac{S^*_i}{S_i} - \beta) T^*(h_i) \\
&\leq \beta Opt + M \underbrace{(\frac{\sum_{i \in X^-} S^*_i}{a'} - \beta card(X^-))}_{\lambda}, \; with\ M = max_{i \in \{g+1, \dots, k\}} (T^*(h_i))
\end{aligned}$$

For the same reason as in Proposition 9, we have $\lambda \leq k' - \beta$. Then, the ratio of $M\bar{A}^{G_2}$ is $r \leq \beta + \frac{k' - \beta}{g+1} = \frac{k + \frac{g}{2^{j_1 - 1}}}{g+1}$. □

Corollary 13. *There is an* $\frac{k + \frac{g}{2^{j_1 - 1}}}{g+1}$*-approximation for r-dRSSP which runs in* $O((k 2^{j_1} \log(m))^g * kn)$*, for any* $g \in \{0 \dots k-1\}$ *and* $j_1 \in \{1, .., \lceil \log(m) \rceil\}$*.*

Notice that the MA^{G_1} algorithm could also be improved with the guess approximation technique. We would get a $\beta(k - g)$ ratio, using an information of size $j_1 + \log(\log(m))$. In our previous work, we also considered another naive algorithm (which simply redistributes a well chosen fraction of the resources given to the guessed heuristics to the others), however it does not bring any significant improvement over the algorithms presented above.

4. Experiments

Some preliminary experiments have been reported in [4] (comparing execution times for MA^{G_1} and MA^{G_2} using the linear cost assumption). Thus we will focus in the following experiments on two axes: studying the impact of the guess approximation on the execution times and returned values, and investigating an example of the non linear case. The complete description (including source codes) of the experiments is available on-line [5].

4.1. *Description of the protocol*

4.1.1. *Inputs of r-dRSSP used for the experiments*

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.

Let us describe here how we constructed our inputs for the experiments. We used a SAT database (SatEx[c]) which gives for a set of 23 heuristics (SAT solvers) and a benchmark of 1303 instances for SAT the CPU execution times (on a single machine) of each heuristics on the 1303 instances of the benchmark. Thus we did not actually run these heuristics, but we used these CPU times to have a realistic matrix cost. 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 and scheduling. These instances are also issued from many challenging benchmarks for SAT which are used in one of the most popular annual SAT competition [d].

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 $heerhugo$, $modoc$, $modoc-2.0$

We define thanks to these $1303*23$ values all the $C(h_i, I_j, m)$, for all i and j. In the linear case, these values are sufficient to entirely define the cost matrix. For the experiments concerning the non-linear case, we have chosen a logarithmic speedup, meaning that we defined $C(h_i, I_j, x) = \frac{C(h_i,I_j,1)}{\log(x)}$. All the considered inputs have the 1303 instances. For the inputs of r-dRSSP with k heuristics, $k < 23$, we randomly chose a subset of heuristics of size k (using an uniform distribution).

4.1.2. *Description of the measures*

There are two types of results: the returned values (the computed cost for solving the considered input) and the execution times. The only difficulty for measuring the returned values concerns MA^{G_1} and $MA^{\bar{G}_1}$. Indeed, recall that these algorithms use an arbitrary subset of g heuristics. Thus, the returned values could depend critically on the chosen subset. To avoid this problem we considered the mean cost of these algorithms over 30 experiments, while choosing randomly (using an uniform distribution) a subset of g heuristics for each experiment and keeping unchanged all the other parameters. The figures show the standard deviations, which were small in all the experiments.

Concerning the execution times, these experiments were done on a AMD

[c]`http://www.lri.fr/~simon/satex/satex.php3`
[d]`http://www.satcompetition.org`

Opteron 246 CPU with 2 cores and 2 GB of memory. For MA^{G_1} and $MA^{\bar{G}_1}$, we considered their mean execution times over the 30 experiments (the standard deviation was negligible).

4.2. *Experiment 1*

The goal of the first experience is to study the impact of the guess approximation technique on the returned values. Thus, the comparison are made between the cost of MA^{G_1}, MA^{G_2} and the one of $MA^{\bar{G}_1}$, $MA^{\bar{G}_2}$ for different values of g. The input of this experiment is constructed with the 23 heuristics, and $m = 100$ machines. The j_1 parameter used in the $MA^{\bar{G}_i}$ algorithm must be between 1 and 6. We decided to only show the results for two representative values of j_1, that is $j_1 = 1$ and $j_1 = 4$. In this case we assumed the linear cost assumption.

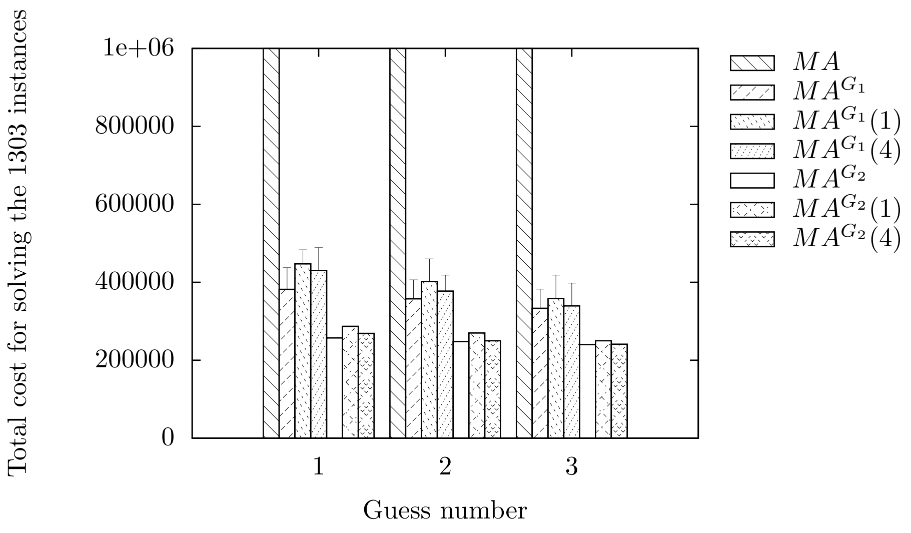

Fig. 1. Discrete Resource Sharing Cost with 23 heuristics and 100 resources

Figure 1 depicts the portfolio cost obtained in this experiment. $MA^{G_i}(x)$ in this picture corresponds to the algorithm $MA^{\bar{G}_i}$ with $j_1 = x$. As one can observe here, the new heuristics proposed are better than MA (which presented an absolute cost of 4253266, or 9.50 times worse than $MA^{G1}(1)$). Moreover, the guess approximation technique (applied in Section 3.5) does not degrade too much the quality of the solutions. Indeed, the ratio between the portfolio cost computed by $MA^{\bar{G}_1}(1)$ and MA^{G_1} for $g = 1$ is equal to 1.17. For MA^{G_2}, this ratio is equal to 1.11.

4.3. *Experiment 2*

In this experiment we are interested in comparisons with the optimal values and computation time. We compare the cost (and the execution time) of MA^{G_1}, MA^{G_2}, $MA^{\bar{G}_1}$ and $MA^{\bar{G}_2}$ with the optimal one for different values of g. Notice that the optimal solution is computed by simple enumeration, leading to a cost in $O(m^k)$. Thereby, the input of this experiment is constructed with only 6 (randomly chosen) heuristics[e], and $m = 50$ machines. We decided to only show the results for two representative values of j_1, that is $j_1 = 1$ and $j_1 = 3$. In this case we also assumed the linear cost assumption.

Figure 2 depicts the portfolio cost obtained in this experiment. One can observe again that it is interesting to restrict the subset of allocation for MA^{G_1} and MA^{G_2}. For instance, when guessing 4 heuristics, the ratio between the portfolio cost of $MA^{\bar{G}_2}(1)$ and the optimal algorithm is equal to 1.3, which is actually lower than the theoretical ratio equal to $\frac{6+4/2}{5} = 1.6$. Notice also that when guessing 4 heuristics with $j_1 = 3$, $MA^{\bar{G}_2}(3)$ finds the optimal portfolio cost. We also illustrate through this experiment the benefit of $MA^{\bar{G}_i}$ algorithms in considering their executions times. In Figure 4, we present the execution times of $MA^{\bar{G}_i}$, MA^{G_i} and the optimal algorithm. The $MA^{\bar{G}_i}$ algorithms in general provide a lower execution time than the other ones. For example, when guessing $g = 4$ heuristics, $MA^{\bar{G}_1}(1)$ is 44 times faster than $MA^{\bar{G}_1}(3)$, which is also 5.3 times faster than $MA^{\bar{G}_1}$. Similarly, $MA^{\bar{G}_2}(1)$ is 201 times faster than $MA^{\bar{G}_2}(3)$, which is also 1.5 times faster than $MA^{\bar{G}_2}$.

4.4. *Experiment 3*

This experiment is exactly the same as Experiment 2, except that we assume here the logarithmic cost function.

Figure 2 depicts the portfolio cost obtained in this experiment. This Figure shows that the relative behavior of the different algorithms does not change from the linear to the logarithmic case. Indeed, the $MA^{\bar{G}_i}$ algorithms compute a solution which is slightly worse than the MA^{G_i} ones. For instance the ratio between the portfolio cost of $MA^{\bar{G}_1}$ and MA^{G_1} for $g = 4$ and $j_1 = 1$ is equal to 1.02. For $MA^{\bar{G}_2}$, this ratio is equal to 1.03. The comparisons with the optimal solution are also tighter: the ratio between the portfolio cost of $MA^{\bar{G}_2}(1)$ (for $g = 4$) and the optimal algorithm is equal to 1.08. These ratios are lower than in the linear case. Indeed, if the cost function is “close to” a constant one, it seems reasonable that the gap between the solutions decreases.

5. Conclusion

In this work we extended our previous results [4] on the linear restricted discrete resource sharing scheduling problem. The main contributions are the complexity

[e]For the sake of reproducibility, the chosen heuristics were: *csat*, $ntab - back2$, *modoc*, *dr*, *ntab*, *zchaff*

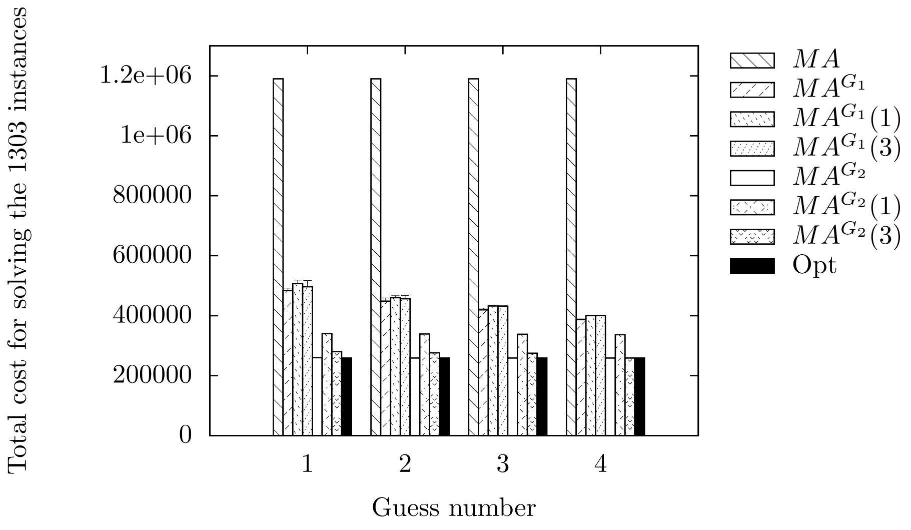

Fig. 2. Discrete Resource Sharing Cost with 6 heuristics and 50 resources (Linear case)

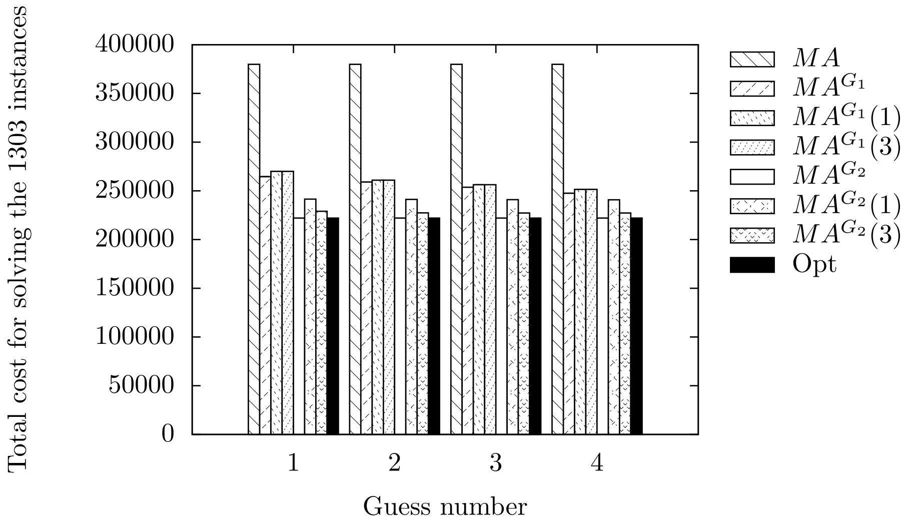

Fig. 3. Discrete Resource Sharing Cost with 6 heuristics and 50 resources (Non-linear case)

proof of lr-dRSSP, the extension (and the improvement using a guess approximation technique [6]) of previous approximation schemes [4] to the non-linear case, the tightness result that shows that the question asked to the oracle was well chosen, and the experiments for these new algorithms. There are many perspectives for continuing this work, namely the proposal of a model and new heuristics for the

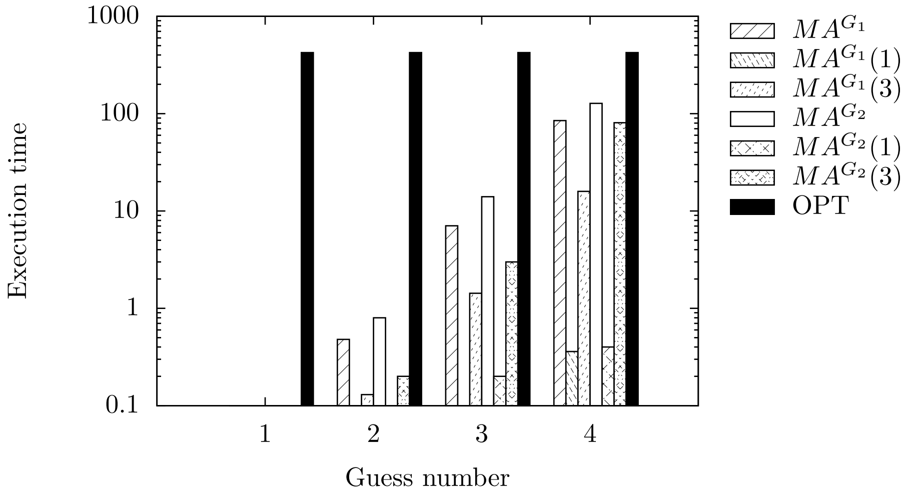

Fig. 4. Execution time with 6 heuristics and 50 resources (Linear case)

case of heterogeneous resources and the study of a mixed problem with both resource sharing and time switching.

References

[1] S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18:283–305, 1997.

[2] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger. STAPL: An adaptive, generic parallel C++ library. *Lecture notes in computer science*, pages 193–208, 2003.

[3] S. Bhowmick, V.Eijkhout, Y.Freund, E. Fuentes, and D. Keyes. Application of machine learning to the selection of sparse linear solvers. `www.tacc.utexas.edu /~eijkhout/Articles/2006-bhowmick.pdf`.

[4] M. Bougeret, P. Dutot, A. Goldman, Y. Ngoko, and D. Trystram. Combining multiple heuristics on discrete resources. In *11th Workshop on Advances in Parallel and Distributed Computational Models APDCM, (IPDPS)*, 2009.

[5] M. Bougeret, P.-F. Dutot, A. Goldman, Y. Ngoko, and D. Trystram. Experiments on approximating the discrete resource sharing scheduling problem. `http://moais.imag.fr/membres/yanik.ngoko/index.php?view=experiment`.

[6] M. Bougeret, P.-F. Dutot, and D. Trystram. The guess approximation technique and its application to the discrete resource sharing scheduling problem. In *Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, 2009.

[7] V. Cicirello. *Boosting Stochastic Problem Solvers Through Online Self-Analysis of Performance.* PhD thesis, Carnegie Mellon University, 2003.

[8] J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, et al. Self-adapting numerical software (SANS) effort. *IBM Journal of Research and Development*, 50(2-3):223–238, 2006.

[9] P. Fraigniaud, C. Gavoille, D. Ilcinkas, and A. Pelc. Distributed computing with

advice: Information sensitivity of graph coloring. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 231–242, 2007.

[10] P. Fraigniaud, D. Ilcinkas, and A. Pelc. Oracle size : a new measure of difficulty for communication tasks. In *25th ACM Symposium on Principles Of Distributed Computing (PODC)*, Denver, Colorado, USA, 2006.

[11] M. Garey, D. Johnson, R. Backhouse, G. von Bochmann, D. Harel, C. van Rijsbergen, J. Hopcroft, J. Ullman, A. Marshall, I. Olkin, et al. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Springer, 1979.

[12] C. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1):43–62, 2001.

[13] B. Huberman, R. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51, 1997.

[14] H. Markowitz. The early history of portfolio theory: 1600-1960. *Financial Analysts Journal*, pages 5–16, 1999.

[15] T. Sayag, S. Fine, and Y. Mansour. Combining multiple heuristics. In Durand and Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *LNCS*, pages 242–253. Springer, 2006.

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

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

[18] L. Simon and P. Chatalic. SATEx: a web-based framework for SAT experimentation. *Electronic Notes in Discrete Mathematics*, 9:129–149, 2001.

[19] M. Streeter, D. Golovin, and S. Smith. Combining multiple heuristics online. In *proceedings of the national conference on artificial intelligence*, volume 22, page 1197. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[20] S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4(4):295–321, 1998.

[21] S. Weerawarana, E. Houstis, J. Rice, A. Joshi, and C. Houstis. PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Transactions on Mathematical Software*, 22(4):447–468, 1996.

[22] H. Yu and L. Rauchwerger. Adaptive reduction parallelization techniques. In *Proceedings of the 14th international conference on Supercomputing*, pages 66–77. ACM New York, NY, USA, 2000.

[23] F. Zheng, Y. Xu, and E. Zhang. How much can lookahead help in online single machine scheduling. *Information Processing Letters*, 106, 2008.