A GRASP for a resource-constrained scheduling problem

Renaud Sirdey[1]*

CEA LIST, Embedded Real Time Systems Lab,
Point Courrier 94, 91191 Gif-sur-Yvette Cedex, France
E-mail: renaud.sirdey@cea.fr
*Corresponding author

Jacques Carlier and Dritan Nace

UMR CNRS Heudiasyc,
Université de Technologie de Compiègne,
Centre de Recherches de Royallieu,
BP 20529, 60205 Compiègne Cedex, France
E-mail: jacques.carlier@hds.utc.fr
E-mail: dritan.nace@hds.utc.fr

Abstract: This paper is devoted to the approximate resolution of a strongly NP-hard real world resource-constrained scheduling problem, which arises in relation to the operability of certain high availability real time distributed systems. We present a fast and pragmatic algorithm based on the GRASP metaheuristic and, building on previous research on exact resolution methods, extensive computational results demonstrating its practical ability to find solutions within a few percents to optimality on a wide spectrum of hard instances.

Keywords: combinatorial optimisation; scheduling; GRASP; distributed systems; OR in telecommunications.

Reference to this paper should be made as follows: Sirdey, R., Carlier, J. and Nace, D. (2010) 'A GRASP for a resource-constrained scheduling problem', *Int. J. Innovative Computing and Applications*, Vol. 2, No. 3, pp.143–149.

Biographical notes: Renaud Sirdey is a Researcher at Commissariat à l'Energie Atomique, the French atomic energy research agency. His main research interests include parallelism, compilation, graph theory and combinatorial optimisation. Prior to that, he held various R&D positions in the telecommunications industry. He received a diplôme d'ingénieur and a PhD in CS from Universitè de Technologie de Compiègne (UTC) as well as an MSc in Applied Maths from Cranfield University (UK). He teaches software engineering at Ecole Nationale Supérieure de Techniques Avancées (Paris) as well as operations research at UTC.

Jacques Carlier is Professor of CS as well as the Head of Doctoral Studies in CS at the Université de Technologie de Compiègne. His main research interests include scheduling, combinatorial optimisation, graph theory and bin-packing. He received his PhD in CS as well as a Doctorat d'Etat both from Paris VI University and is an alumnus of Ecole Normale Supérieure de Cachan.

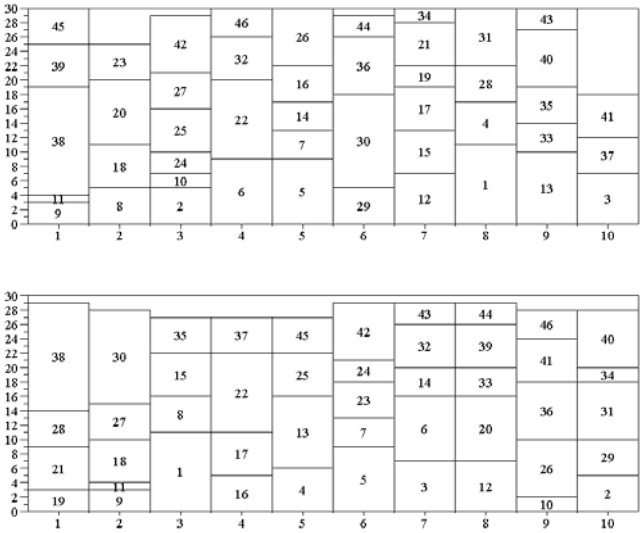
Dritan Nace is Professor of CS at the Université de Technologie de Compiégne. His main research interests include networks optimisation, robustness, linear programming and combinatorial optimisation. He received his PhD in CS as well as his Habilitation à Diriger les Recherches both from the Université de Technologie de Compiègne.

1 Introduction

In this paper, we present a GRASP for the *process move programming* (PMP) problem. This problem arises in relation to the operability of certain high-availability distributed switching systems. For example, Sirdey et al. (2003), consider a telecom switch managing radio cells on a set of call processing modules, hereafter referred to as *processors*, of finite capacity in terms of erlangs, CPU, memory, ports, etc.; each radio cell being managed by a dedicated process running on some processor. During network operation, some cells may be dynamically added, modified (transreceivers may be added or removed) or removed, potentially leading to unsatisfactory resource utilisation in the system. This issue is addressed by first obtaining a better system configuration and by subsequently reconfiguring the system, without violation of the capacity constraints on the processors.

Figure 1 provides an example of an instance of the PMP problem for a system with ten processors, one resource and 46 processes. The capacity of each of the processors is equal to 30 and the sum of the consumptions of the processes is 281. The top and bottom figures respectively represent the initial and the final system states. For example, process number 23 must be moved from processor 2 to processor 6.

Figure 1 Example of an instance of the PMP problem



As discussed in Section 3, the PMP problem has already been tackled with both combinatorial (branch-and-bound) and polyhedral (branch-and-cut) exact resolution methods as well as computationally intensive heuristic approaches such as simulated annealing. Still, for real-time constraints related to the industrial context of the present work, there is a need for fast, pragmatic approximate resolution algorithms with reasonable implementation complexity. Our motivation for using the GRASP paradigm to design such an algorithm is based on software engineering considerations. Indeed, the GRASP paradigm allows to achieve an interesting trade-off between the following criteria: implementation complexity (which should be understood as a measure of how difficult it is to write and, more importantly, to subsequently maintain the resulting software[2]), calculation time and performance (in terms of quality of the provided solutions). Usually, the operations research literature focuses on the two latter criteria and largely omits the first one which in many real world situations is at least as important [see Sirdey (2006) for a more thorough discussion]. A polyhedral exact resolution method, e.g., requires fairly advanced mathematical skills to be understood (a prerequisite for maintaining the software) as well as high computation times to provide optimum or provably high-quality solutions. A simulated annealing-based algorithm, still e.g., usually achieves a very different trade-off: understanding and programming it is simple however it still requires fairly important computation times to provide reasonably (and certainly not provably) good solutions. The GRASP approach to combinatorial optimisation problems attempts (in the authors' opinion) to achieve yet another trade-off with an implementation complexity slightly higher that an SA-based algorithm but an ability to provide reasonably (still not provably) good solutions at significantly lower computational costs. This makes GRASP a paradigm of choice for designing an optimisation algorithm for a real world problem with real world operational and software engineering constraints.

This paper is organised as follows. First, Section 2 provides a formal statement of the problem and Section 3 succinctly discusses previous work on the PMP problem. Then, Section 4 provides background on GRASP and Section 5 presents our algorithm. Lastly, in Section 6, we provide extensive computational results which demonstrate the practical relevance of the approach.

2 Formal problem statement

Let us consider a distributed system composed of a set U of *processors*, each processor offering an amount $C_u \in \mathbb{N}$ of a given resource. We are also given a set P of applications, hereafter referred to as *processes*, which consume the resources offered by the processors. The set P is sometimes referred to as the *payload* of the system. For each process $p \in P$, $w_p \in \mathbb{N}$ denotes the amount of resource which is consumed by process p. Note that neither C_u nor w_p vary with time.

An *admissible state* for the system is defined as a mapping $f : P \to U \cup \{u_\infty\}$, where u_∞ is a dummy processor having infinite capacity, such that for all $u \in U$ we have:

$$\sum_{p \in P(u;f)} w_p \leq C_u, \tag{1}$$

where $P(u;f) = \{p \in P : f(p) = u\}$. The processes in $\overline{P}(f) = P(u_\infty; f)$ are not instantiated and, when this set is non-empty, the system is in *degraded mode*.

An instance of the PMP problem is then specified by two arbitrary system states f_i and f_t respectively referred to as the *initial system state* and the *final system state* or, for short, the *initial state* and the *final state*.[3]

A process may be moved from one processor to another in two different ways: either it is migrated, in which case it consumes resources on both processors for the duration of the migration and this operation has virtually no impact on service, or it is *interrupted*, that is removed from the first processor and later restarted on the other one. Of course, this latter operation has an impact on service. Additionally, it is required that the capacity constraints (1) are always satisfied during the reconfiguration and that a process is moved (i.e., migrated or interrupted) at most once. This latest constraint is motivated by the fact that a process migration is far from being a lightweight operation [for reasons related to distributed data consistency which are out of the scope of this paper, see e.g., Jalote (1994)] and, as a consequence, it is desirable to avoid processes hopping around processors.

Throughout this paper, when it is said that a move is *interrupted*, it is meant that the process associated to the

move is interrupted. This slightly abusive terminology significantly lightens our discourse.

For each processor u, a process p in $P(u;f_i)\setminus P(u;f_t)$ must be moved from u to $f_t(p)$. Let M denote the set of process moves thus induced by the initial and final states. Then for each $m \in M, w_m, s_m$ and t_m respectively denote the amount of resource consumed by the process moved by m, the processor from which the process is moved that is the *source* of the move and the processor to which the process is moved that is the *target* of the move. Lastly, $S(u)=\{m \in M : s_m = u\}$ and $T(u)=\{m \in M : t_m = u\}$.

A pair (I,σ), where $I \subseteq M$ and $\sigma : M\setminus I \to \{1,...,|M\setminus I|\}$ is a bijection, defines an admissible *process move programme*, if provided that the moves in I are interrupted (for operational reasons, the interruptions are performed at the beginning) the other moves can be performed according to σ without inducing any violation of the capacity constraints (1). Formally, (I,σ) is an admissible programme if for all $m \in M \setminus I$ we have:

$$w_m \le K_{t_m} + \sum_{\substack{m' \in I \\ s_{m'}=t_m}} w_{m'} + \sum_{\substack{m' \in S(t_m)\setminus I \\ \sigma(m')<\sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m)\setminus I \\ \sigma(m')<\sigma(m)}} w_{m'}, \quad (2)$$

where $K_u = C_u - \Sigma_{p \in P(u;f_i)} w_p$ denotes the remaining capacity on processor u in the initial state, thereby guaranteeing that the intermediate states are admissible.

Also note that because the final state is admissible, we have for each processor $u \in U$.

$$K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u)} w_m \ge 0. \quad (3)$$

Let c_m denote the cost of interrupting $m \in M$, the PMP problem then formally consists, given a set of moves, in finding a pair (I,σ) such that $c(I)=\sum_{m \in I} c_m$ is minimum.

3 Related work

The PMP problem is now relatively well studied from an exact resolution perspective. Sirdey et al. (2007) have shown that the PMP problem is strongly NP-hard, exhibited some polynomially solvable special cases (the most notable one being $|R|=1$ and $w_m = w$ for all $m \in M$) as well as proposed a 'combinatorial' branch-and-bound algorithm for the general case (an extensive literature survey is also provided in that paper). Additionally, Sirdey and Kerivin (2006) have studied the problem from the point of view of polyhedral combinatorics, leading to an exact resolution branch-and-cut algorithm. In terms of approximate resolution, a computionally intensive, although quite theoretically sound, simulated annealing-based algorithm has been proposed by Sirdey et al. (2009). Still, as already stated in the introduction, for operational reasons related to the industrial context of this work, there is a need for fast, pragmatic approximate resolution algorithms. Thus, in this paper, we present such an algorithm based on the GRASP metaheuristic.

4 GRASP in a nutshell

GRASP[4] is an approximate resolution algorithm design paradigm introduced in the nineties by Feo and Resende (1989, 1995). This paradigm is interesting for it combines two fairly natural techniques for heuristically dealing with hard combinatorial problems: greedy algorithms and local search.

A GRASP is a multi-start heuristic, at each step a *randomised greedy algorithm* is used to build an admissible solution which neighbourhood is explored using a *local search procedure*. The best of these thus obtained solutions is output by the algorithm.

Given a combinatorial optimisation problem, a greedy algorithm is an algorithm which iteratively builds an admissible solution by, at each iteration, making the decision resulting in the best objective function improvement, each such decisions being definitive. In general, greedy algorithms do not perform particularly well and randomisation is a valuable tool in order to improve their performances, as Karp (1991) puts it:

> "Often, the introduction of randomization suffices to convert a simple and naive deterministic algorithm with bad worst-case behavior into a randomized algorithm that performs well with high probability on every possible input."

The randomisation scheme usually employed in GRASP implementations has been proposed by Hart and Shogan (1987). At each iteration, the algorithm considers the best and worst objective function improvements, say γ^+ and γ^- respectively, and make a decision drawn uniformly from the set of decisions which improve the objective function by at least $\gamma^+ + \alpha(\gamma^- - \gamma^+)$ where $\alpha \in [0,1]$. The parameter controls the amount of greediness: when $\alpha = 0$ the algorithm systematically makes the decisions which best improve the objective function hence builds greedy solutions, whereas when $\alpha = 1$ the algorithm builds random ones.

In general, there is no guarantee for solutions built using a randomised greedy algorithm to be locally optimal with respect to a given neighbourhood structure. Hence, it is relevant to start a local search procedure at each or some (depending on the computational cost of the procedure) of the solutions provided by the construction phase.

At present, GRASP seems to emerge as one of the leading paradigms for designing efficient approximate resolution algorithms for hard combinatorial optimisation problems. An evidence of this being that it has been successfully applied to a wide range of such problems; see the survey papers by Resende (1998) as well as by Resende and Ribeiro (2003).

5 Application to the PMP problem

This section outlines our GRASP for the PMP problem. We thus present the construction and local search phases as well

as detail how the two are glued together to obtain a proper GRASP.

5.1 Construction phase

Let I denote the (initially empty) set of moves which are interrupted and R denote the (initially equal to M) set of moves which are yet neither ordered nor scheduled. Also, let σ denote an ordering of the moves in $M \setminus (I \cup R)$ which is admissible under the assumption that the moves in I are interrupted and that the moves in R are not performed i.e., that the associated process simply remains on the source processor.

At each iteration, we then proceed as follows.

Let $X \subseteq R$ denote the set of moves which can be inserted in σ without jeopardising its admissibility (as defined above). Until this set is non-empty, a move is drawn uniformly from $\{m \in R : c_m \leq \alpha c_{\max}(R) + (1-\alpha) c_{\min}(R)\}$ removed from R and added to I. At that point, X is non-empty and a move is then drawn uniformly from $\{m \in X : c_m \geq \alpha c_{\max}(X) + (1-\alpha) c_{\min}(X)\}$ removed from R and inserted as early as possible into σ. Lastly, the set I is minimalised. Let $Y \subseteq I$ be such that the new ordering σ remains admissible if any move $m \in Y$ is neither interrupted nor performed. Then a move is drawn uniformly from $\{m \in Y : c_m \geq \alpha c_{\max}(Y) + (1-\alpha) c_{\min}(Y)\}$ removed from I and added to R, until Y ends up being empty.

The above scheme is repeated until R is empty.

The ability for moves to switch back and forth between R and I is undoubtfully an important feature of the above algorithm. Indeed, interruption decisions are regularly challenged and potentially backtracked on. As an example, it leaves the possibility for one or more interruption decisions to eclipse a preceding one, therefore giving a second chance to the move associated to the latter.

5.2 Local search phase

The construction phase is completed by a local search phase based on a light-weight exchange strategy between $M \setminus I$ and I.

Indeed, each iteration of the local search algorithm consists, for each scheduled move (i.e., each move in $M \setminus I$), say m, in checking whether interrupting it allows to insert a move $m' \in I$, with a cost higher than m, into σ. As soon as such a pair of moves is found, a new solution is generated in which m is interrupted and m' is performed as early as possible.

This process is repeated until no such pair exists, hence, until local optimality is established with respect to this simple neighbourhood structure.

5.3 Putting it all together

We have integrated the above construction and local search phases following the GRASP paradigm so as to obtain a fast heuristic for the PMP problem[5]. The fastness requirement implied a small number of iterations, $|M| \log |M|$ was chosen, as well as guided our choice of a strategy for the variation of α.

Indeed, there are three main such strategies: self tune α according to the reactive procedure of Prais and Ribeiro (2000), draw α uniformly from [0, 1] or draw from [0, 1] according to another probability distribution. Computational experiments reported by Resende and Ribeiro (2003) hint that when computation time is an issue, the uniform strategy offers the most appropriate trade-off, in particular compared to the reactive strategy which appears superior to the others at the cost of longer computation time.

Hence, our GRASP algorithm reduces to repeating $|M| \log |M|$ times the construction phase of Section 5.1 followed by the local search phase of Section 5.2, each time with a value of α uniformly drawn from [0, 1]. The best of these thus obtained solutions is then returned.

6 Computational experiments

It should be emphasised that, following the discussion in the introduction, the purpose of this paper and of the experiments reported in this section is *not* to conclude on the relative calculation cost-to-performance ratio between our GRASP and other methods. Our goal is merely to illustrate the fact that our GRASP achieves the software engineering trade-off which was committed to in the introduction. As an algorithm which can be specified in a few pages, the implementation complexity of our GRASP is clearly under control. Thus, in this section, we further report on computational experiments carried out so as to assess the ability of our procedure to produce reasonably good solutions at low computational cost.

These experiments have been performed on a Sun Ultra 10 workstation with a 440 MHz Sparc microprocessor, 512 Mo of memory and the Solaris 5.8 operating system. For reproductibility purpose, the instances used in the present experiments can be made available upon email request to the corresponding author.

6.1 Instance generation

Given U the set of processors, C the processor capacity and W an upper bound on the process consumption, an instance is generated as follows.

First, the set of processes is built by drawing consumptions uniformly in $\{1,...,W\}$ until $\sum_{p \in P} w_p \geq C|U|$. The initial state, f_i, is then generated by randomly assigning the processes to the processors: the processor to which a process is assigned is drawn uniformly from the set of processors which remaining capacity is sufficient (note that not all processes necessarily end up assigned to a processor). The final state, f_t, is built in the very same way to the exception that only the processes which are assigned to a processor in the initial state are considered. An instance is considered valid only if all the processes assigned to a

processor in the initial state are also assigned to a processor in the final state. Invalid instances are discarded and the construction process is repeated until a valid instance is obtained (the rejection rate depends on the parameters, as an example, coarse estimates for $|U|=10$, $C=100$ as well as $W=10$ and $W=50$ respectively are 29% and 41%). The set of moves is then built as explained in Section 2.

It should be emphasised that the above scheme generates instances for which the capacity constraints are extremely tight, instances which can be expected to be hard and, in particular, significantly harder than those occurring in practice. As an example, for $|U|=10$, $C=100$ and $W=10$ only 1.28% of free capacity remains, on average, on each of the processors. However, for the system to which this work is to be applied (see Sirdey et al., 2003) the maximum theoretical load of a processor ranges (non-linearly) from at most 50% (for a system with two processors) to at most around 93% (for a system with 14 processors, which is the maximum). This is so because some spare capacity is provisioned for fault tolerance purpose and this spare capacity is spread among all the processors. Additionally, it should be stressed that the system carries at most 100 processes and that a preprocessing technique, based on the fact that the properties of a system state are invariant by a permutation of the processors, is used to decrease the number of moves by around 25% on average. It turned out that our GRASP was able to solve virtually all practical instances and that as a consequence, we had to consider more aggressive instance generation schemes, such as the above, in order to fairly evaluate the performances of the algorithm.

Lastly, we have supposed that $c_m = w_m$, which is quite natural for our application as it is reasonable to assume that the amount of service provided by a process is proportional to the amount of resources it consumes.

6.2 Computational results

In order to reasonably explore the (practically relevant part of the) problem space we have used the scheme of Section 6.1 to generate a set of ten instances for each $|U| \in \{2,...,14\}$,[6] each $W \in \{10,20,...,90,100\}$ and $C=100$. Hence, a total of 1300 instances, amongst which only 1020 were considered of non-trivial size (up to 254 moves) and used in our experiments.

In fact, the instance base is the same that we used in order to assess the practical relevance of the branch-and-bound algorithm presented in Sirdey et al. (2007), the advantage being that the value of an optimum solution is known for many of these instances. This instance base was also used for the empirical evaluation of our simulated annealing algorithm (Sirdey et al., 2009).

For each of the above ten instances sets, Table 1 indicates the average problem size (i.e., the average number of moves), denoted $|\overline{M}|$, as well as an upper bound on the average optimality gap, denoted $\overline{d}$, which we were able to (manually) prove using either the output of our branch-and-bound algorithm (Sirdey et al., 2007), i.e., which were solved to optimality by that algorithm, or, otherwise, the polyhedral lower bound introduced by Sirdey and Kerivin (2006).

Despite slightly disappointing results on instances located within the small number of moves (say less than 20) and small number of processors (say less than seven) area (we shall come back to this), Table 1 illustrates the fairly good performances of our GRASP which achieves an overall average optimality gap of 1.68% (despite incomplete knowledge of the optimum values leading to some overestimation of that number). Indeed, in the eight to 14 processors area, the average upper bound obtained for the optimality gap is always below 4% (still despite incomplete knowledge of the optimum values). It turns out that when W is high enough (typically above 70) the instances are relatively small and, as a consequence, an optimum solution is known for many of them. Hence, the average optimality gap is relatively reliably estimated in that area and the good performances of the algorithm are illustrated. When W is low enough (typically below 30), small costs solutions almost always exist and can be found by the algorithm as illustrated by the small average optimality gaps observed in the corresponding area. In between these two areas, the average optimality gap seems greater. This is nevertheless most likely explained by the fact that in that area there are less instances for which an optimum solution is known, hence, the gap estimation relies more on the polyhedral bound and this potentially leads to significant overestimation.

Returning to the small number of moves and small number of processors area, it appears that the algorithm is able to solve most instances to optimality but, from time to time, fails to do so and produces solutions quite far from an optimal solution (gaps of up to 16% were observed). It turns out that this is imputable to the fact that the construction phase does not consider interrupting moves admissible for insertion in the partial ordering σ. Indeed, we have shown in Sirdey et al. (2009) that an algorithm which does not consider such decisions may simply systematically miss all optimum solutions on certain pathological instances. The problem is that modifying the algorithm so as to take these decisions into account (which requires alternating overall emptying of R and minimilisation of I, versus step-by-step, during the construction phase) allows to satisfactorily solve the small pathological instances but translates into a significant performance degradation on the remaining ones. A price which we are not prepared to pay. As the pathological instances are both very small, few in numbers and accessible in virtually no time to our branch-and-bound algorithm, we consider their existence only a minor nuisance.

Table 1 Average instance size, denoted $\overline{|M|}$, and an upper bound on the average optimality gap, denoted $\overline{d}$, for each of the ten instances sets generated

$W\backslash\|U\|$	*2*		*3*		*4*		*5*		*6*		*7*		*8*	
	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$
10	17.3	0.64	37.3	2.14	54.4	1.26	73.1	1.59	86.8	1.02	110.1	0.95	125.8	0.88
20	8.2	2.01	19.5	0.69	26.7	1.33	35.0	2.78	46.7	1.48	56.5	1.53	64.1	1.13
30	6.4	2.40	12.9	4.06	19.5	2.18	23.9	2.58	30.4	2.37	37.2	2.74	44.6	2.42
40			9.9	1.73	12.5	2.97	19.3	2.64	22.9	0.95	28.1	2.72	33.9	2.33
50					10.6	2.86	12.9	1.18	19.5	1.58	22.0	2.30	25.9	2.16
60							13.2	2.82	14.6	3.12	18.1	4.30	21.4	3.05
70									13.3	4.99	15.2	0.00	18.6	0.73
80											11.9	0.00	15.4	0.00
90													12.9	0.00

$W\backslash\|U\|$	*9*		*10*		*11*		*12*		*13*		*14*	
	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$	$\overline{\|M\|}$	$\overline{d}$
10	150.1	0.99	159.2	1.02	179.5	0.68	198.5	1.10	215.8	0.74	237.6	0.74
20	75.6	1.26	82.1	0.88	92.5	1.16	102.6	0.70	111.1	1.43	122.4	1.92
30	47.2	2.22	56.7	2.11	64.6	2.00	71.2	1.01	77.5	2.40	80.6	1.50
40	37.3	2.20	45.7	2.48	48.0	2.36	51.6	2.03	56.8	2.29	58.6	1.51
50	30.1	3.24	33.5	3.42	37.8	3.17	41.8	2.45	43.7	2.49	53.0	4.09
60	25.8	2.00	29.5	2.42	29.2	1.94	31.8	1.65	35.3	3.24	40.8	3.33
70	22.1	1.70	23.2	0.85	25.7	2.01	28.1	1.90	32.2	2.94	36.3	2.77
80	17.3	0.00	19.0	0.92	21.2	0.32	25.1	0.31	25.5	0.51	28.4	0.00
90	16.3	0.61	18.9	1.01	20.8	0.61	23.6	0.62	22.8	0.88	26.4	0.66
100	12.8	0.00	15.8	0.29	17.9	0.00	18.2	0.08	19.6	0.72	22.7	0.56

Note: $|U|$ denotes the number of processors and W the maximum process consumption (see text).

7 Conclusions

In this paper, we have proposed a fast GRASP-based heuristic for the PMP problem, a strongly NP-hard scheduling problem which consists, starting from an arbitrary initial process distribution on the processors of a distributed system, in finding the least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint is that the capacity of the processors must not be exceeded during the reconfiguration. This problem has applications in the design of high availability real-time distributed switching systems such as the one discussed in Sirdey et al. (2003).

In particular, we have introduced a randomised greedy algorithm based on validity preserving insertions of moves within a valid partial ordering. This algorithm has the desirable property of challenging and potentially backtracking on interruption decisions at each step. A GRASP algorithm has then been designed by complementing the aforementioned algorithm with a lightweight local search scheme.

Additionally, building on previous research on exact resolution procedures (Sirdey et al., 2007; Sirdey and Kerivin, 2006), we have reported on extensive computational experiments illustrating the ability of our algorithm to produce solutions within a few percents to optimality on a wide spectrum of 'hard' instances, hard in terms of both size and tightness of the capacity constraints. This is despite the presence of a few small pathological instances due to the fact that certain decisions are deliberately not considered during the construction phase. However, due to their small size, these pathological instances are easily accessible to exact resolution procedures.

Lastly, it should be emphasised that as far as solving the PMP problem in the industrial setting in which it originally cropped up, the present GRASP-based heuristic has been considered the most suitable for, as intended, it achieves an appropriate software engineering trade-off in that context: low implementation complexity (to meet the software maintainability constraints), low computational cost (to meet the real-time constraints) and good enough performances. In particular, with respect to computational cost, this algorithm is around one order of magnitude faster than the simulated annealing scheme of Sirdey et al. (2009) and several orders of magnitude faster than the branch-and-bound algorithm of Sirdey et al. (2007).[7] For these reasons, the present GRASP algorithm is *intrinsically*

practically superior to the other approaches reported in the literature, *with respect to the present industrial context*, and it thus turned out that this was the algorithm finally implemented in the target switching systems, as a result of a collegial engineering decision in which the first author took part.

References

Feo, T.A. and Resende, M.G.C. (1989) 'A probabilistic heuristic for a computationally difficult set covering problem', *Operations Research Letters*, Vol. 8, pp.67–71.

Feo, T.A. and Resende, M.G.C. (1995) 'Greedy randomized adaptive search procedure', *Journal of Global Optimization*, Vol. 6, pp.109–133.

Hart, J.P. and Shogan, A.W. (1987) 'Semi-greedy heuristics: an empirical study', *Operations Research Letters*, Vol. 6, pp.107–114.

Jalote, P. (1994) 'Fault tolerance in distributed systems', *Distributed Systems*, Prentice Hall.

Karp, R.M. (1991) 'An introduction to randomized algorithms', *Discrete Applied Mathematics*, Vol. 34, pp.165–201.

Prais, M. and Ribeiro, C.C. (2000) 'Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment', *INFORMS Journal on Computing*, Vol. 12, pp.164–176.

Resende, M.G.C. (1998) 'Greedy randomized adaptive local search procedures (GRASP)', Technical Report 98.41.1, AT&T Labs Research.

Resende, M.G.C. and Ribeiro, C.C. (2003) 'Greedy randomized adaptive search procedures', in *Handbook of Metaheuristics, Volume 57 of International Series in Operations Research & Management*, Springer.

Sirdey, R. (2006) 'Combinatorial optimization problems in wireless switch design', *4OR*, Vol. 5, pp.319–33.

Sirdey, R. and Kerivin, H. (2006) 'A branch-and-cut algorithm for a resource-constrained scheduling problem', *RAIRO – Operations Research*, Vol. 41, pp.235–251.

Sirdey, R., Carlier, J. and Nace, D. (2009) 'Approximate resolution of a resource-constrained scheduling problem', *Journal of Heuristics*, Vol. 15, pp.1–17.

Sirdey, R., Carlier, J., Kerivin, H. and Nace, D. (2007) 'On a resource-constrained scheduling problem with application to distributed systems reconfiguration', *European Journal of Operational Research*, Vol. 183, pp.546–563.

Sirdey, R., Plainfossé, D. and Gauthier, J-P. (2003) 'A practical approach to combinatorial optimization problems encountered in the design of a high availability distributed system', in *Proceedings of the International Network Optimization Conference*, pp.532–539.

Notes

1 This research was done while Renaud Sirdey was a System Architect within Nortel GSM Access R&D, Châteaufort, France.

2 Which, more often than not, is done by individuals other than the initial designer.

3 Throughout the rest of this paper, it is assumed that $\overline{P}(f_i) = \overline{P}(f_t) = \emptyset$. When this is not the case the processes in $\overline{P}(f_t) \setminus \overline{P}(f_i)$ should be stopped before the reconfiguration, hence, some resources are freed, the processes in $\overline{P}(f_i) \setminus \overline{P}(f_t)$ should be started after the reconfiguration and the processes in $\overline{P}(f_i) \cap \overline{P}(f_t)$ are irrelevant.

4 Greedy randomised adaptive search procedure.

5 A 'good' computationally intensive heuristic is provided by our simulated annealing scheme (Sirdey et al., 2009).

6 The choice for the values of $|U|$ is motivated by the fact that the system to which this work is to be applied contains at least two and at most 14 processors (Sirdey et al., 2003).

7 Furthermore, this algorithm exhibits widely disparate computation times, as exact procedures for NP-hard problems usually do. Thus, provided the real time context of our application and its implementation cost which is higher than that of our GRASP, its practical relevance is limited to providing test instances used for the validation of approximate resolution an algorithm (which still makes it a precious practical tool).