J Heuristics (2009) 15: 1–17
DOI 10.1007/s10732-007-9052-0

Approximate solution of a resource-constrained scheduling problem

Renaud Sirdey · Jacques Carlier · Dritan Nace

Received: 5 January 2006 / Revised: 12 December 2006 / Accepted: 29 March 2007 /
Published online: 31 October 2007

Abstract This paper is devoted to the approximate solution of a strongly NP-hard resource-constrained scheduling problem which arises in relation to the operability of certain high availability real time distributed systems. We present an algorithm based on the simulated annealing metaheuristic and, building on previous research on exact solution methods, extensive computational results demonstrating its practical ability to produce acceptable solutions, in a precisely defined sense. Additionally, our experiments are in remarkable agreement with certain theoretical properties of our simulated annealing scheme. The paper concludes with a short discussion on further research.

Keywords Combinatorial optimization · Scheduling · Simulated annealing · Distributed systems · OR in telecommunications

1 Introduction

In this paper, we present a simulated annealing-based approximate solution algorithm for the *Process Move Programming* (PMP) problem. This problem arises in

This research was supported in part by Association Nationale de la Recherche Technique grant CIFRE-121/2004.

R. Sirdey (✉)
Service d'architecture BSC, Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France
e-mail: renauds@nortel.com

R. Sirdey · J. Carlier · D. Nace
UMR CNRS Heudiasyc, Université de Technologie de Compiègne,
Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France

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.

We now proceed with a formal definition of the problem.

Let us consider a distributed system composed of a set U of *processors* and let R denote the set of *resources* they offer. For each processor $u \in U$ and each resource $r \in R$, $C_{u,r} \in \mathbb{N}$ denotes the amount of resource r offered by processor u. 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 $p \in P$ and each resource $r \in R$, $w_{p,r} \in \mathbb{N}$ denotes the amount of resource r which is consumed by process p. Note that neither $C_{u,r}$ nor $w_{p,r}$ vary with time. Also, when $|R| = 1$, $C_{u,r}$ and $w_{p,r}$ are respectively denoted C_u and w_p (this principle is applied to other quantities throughout this paper).

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

$$\sum_{p \in P(u;f)} w_{p,r} \leq C_{u,r}, \tag{1}$$

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

An instance of the *Process Move Programming* (PMP) problem is then specified by two arbitrary system states f_i and f_t and, roughly speaking, consists in, starting from state f_i, finding the least disruptive sequence of operations at the end of which the system is in state f_t. The two aforementioned system states are respectively referred to as the *initial system state* and the *final system state* or, for short, the *initial state* and the *final state*.[1]

Figure 1 provides an example of an instance of the PMP problem for a system with 10 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.

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

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

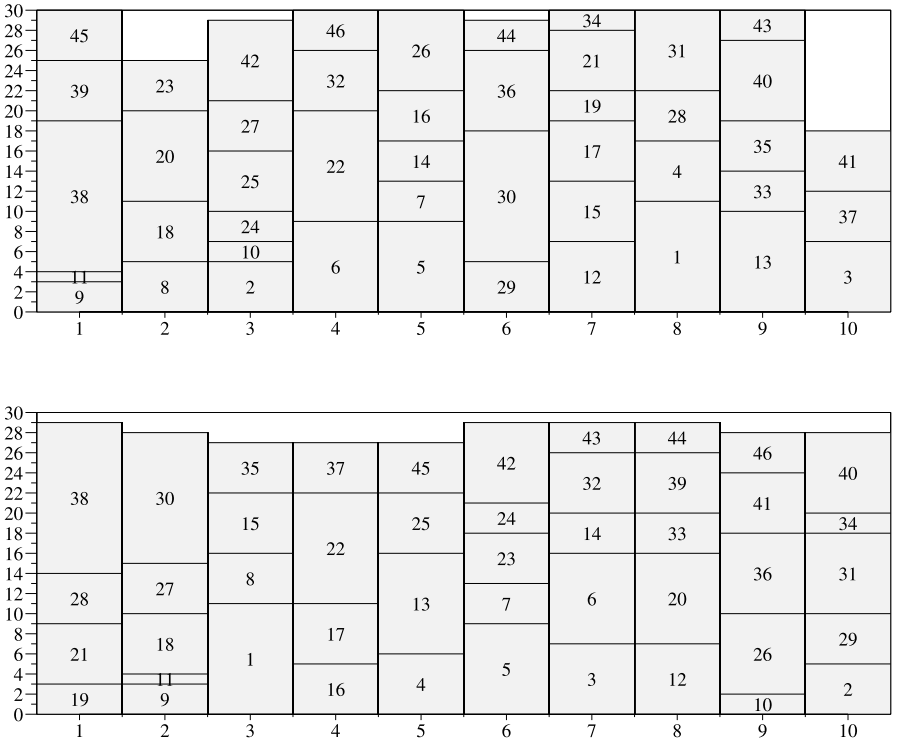

Fig. 1 Example of an instance of the PMP problem

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 (Eq. 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), 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. Additionally, it is now assumed that $|R| = 1$, unless otherwise stated.

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

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

where $K_u = C_u - \sum_{p \in P(u; f_i)} w_p$, thereby guaranteeing that the intermediate states are admissible.

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

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

Let c_m denote the cost of interrupting 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.

In Sirdey et al. (2005a) we have shown that the PMP problem is strongly NP-hard (even for a system with only two processors and only one resource), exhibited some polynomially solvable special cases (the most notable being $|R| = 1$ and $w_m = \text{const}$ for all $m \in M$) as well as proposed a branch-and-bound algorithm for the general case. This paper focuses on an approximate solution algorithm based on the simulated annealing metaheuristic. Section 2 is dedicated to the theoretical considerations at the basis of our algorithm: we introduce the notion of (α, β)-acceptable solution, where β is a measure of distance to optimality and α is the probability that it is achieved, and use the Markovian theory underlying the homogeneous simulated annealing algorithm to derive conditions under which such solutions may be produced. Based on these considerations, a simulated annealing-based pseudopolynomial time approximation algorithm for the PMP problem is presented in Sect. 3. Building on results obtained using the aforementioned branch-and-bound algorithm, we provide in Sect. 4 extensive computational results demonstrating the practical relevance of the method in the special case where $c_m = w_m$ (this variant still is strongly NP-hard).

2 SA-based differential approximation

Simulated annealing is a popular approximate solution algorithm design paradigm independently introduced in the mid eighties by Kirkpatrick et al. (1983) and Cerny (1985). The main advantages of this paradigm are that it leads to relatively simple algorithms and that it is reasonably well understood from a theoretical point of view.

Throughout this section, we consider a combinatorial optimization problem which consists, given a finite set $\Omega = \{\omega_1, \ldots, \omega_N\}$ and an *objective function* $c : \Omega \longrightarrow \{e_1, \ldots, e_P\}$, in looking for an element $\omega^\star \in \Omega$ such that $e_1 = c(\omega^\star) \leq c(\omega) \leq e_P$ for all $\omega \in \Omega$. Also, a *neighbourhood function* $V : \Omega \longrightarrow 2^\Omega$ is given.

Algorithm 1 General form of the simulated annealing algorithm

Do $T \leftarrow T_0$.
Choose ω uniformly in Ω and do $\omega^\star \leftarrow \omega$.
While the stopping criterion is not satisfied do:
  Choose ω' uniformly in $V(\omega)$.
  Choose u uniformly in $[0, 1]$.
  If $u \leq e^{-\frac{c(\omega')-c(\omega)}{T}}$ then[a]
    Do $\omega \leftarrow \omega'$.
    If $c(\omega) < c(\omega^\star)$ then do $\omega^\star \leftarrow \omega$.
  End.
  $T \leftarrow f(T)$.
End.

[a]Note that $e^{-\frac{c(\omega')-c(\omega)}{T}} \geq 1$ when $c(\omega') \leq c(\omega)$

Algorithm 1 states the simulated annealing algorithm in a fairly general form. T_0 is the *initial temperature* and f (usually[2]) is a nonincreasing function referred to as the *cooling schedule*.

For background on the simulated annealing method, the reader is referred to the seminal treatise by van Laarhoven and Aarts (1987).

2.1 Markovian model of the SA algorithm

As noted by Aarts and van Laarhoven (1985) as well as by Lundy and Mees (1986) the behavior of the simulated annealing algorithm at temperature T can be described by means of a finite homogeneous Markov chain[3] with *transition matrix*

$$A_{ij}(T) = \begin{cases} 0 & \text{if } \omega_j \notin V(\omega_i), \\ \frac{1}{|V(\omega_i)|} & \text{if } \omega_j \in V(\omega_i) \text{ and } c(\omega_j) \leq c(\omega_i), \\ \frac{e^{\frac{c(\omega_i)-c(\omega_j)}{T}}}{|V(\omega_i)|} & \text{if } \omega_j \in V(\omega_i) \text{ and } c(\omega_j) > c(\omega_i), \\ 1 - \sum_{j:\omega_j \in V(\omega_i)} A_{ij}(T) & \text{if } i = j. \end{cases}$$

Under the assumption that it is both *regular* (i.e., the directed graph induced by the neighbourhood function is strongly connected) and *aperiodic*, the above chain admits a unique *stationary distribution* given by (recall that $N = |\Omega|$)

$$\pi_i^{(\infty)}(T) = \frac{e^{-\frac{c(\omega_i)}{T}}}{\sum_{j=1}^{N} e^{-\frac{c(\omega_j)}{T}}}. \tag{4}$$

[2]A few authors such as Hajek and Sasaki (1989) and Möbius et al. (1997) consider cooling schedules in which the temperature is allowed to increase.

[3]The reader unfamiliar with the theory of finite Markov chains is referred to Kemeny and Snell (1960).

Also,

$$\lim_{T\to 0} \pi_i^{(\infty)}(T) = \lim_{T\to 0} \frac{1}{\sum_{j=1}^{N} e^{\frac{c(\omega_i)-c(\omega_j)}{T}}} = \begin{cases} 0 & \text{if } c(\omega_i) > e_1, \\ \frac{1}{|\Omega^\star|} & \text{otherwise.} \end{cases}$$

2.2 Probabilistic performance guarantees

We now take the differential approximation theory point of view.[4]

Definition 1 A solution $\omega \in \Omega$ is β-acceptable if

$$c(\omega) \leq e_1 + \beta(e_P - e_1)$$

and (α, β)-acceptable if

$$\text{Prob}(c(\omega) \leq e_1 + \beta(e_P - e_1)) \geq \alpha,$$

where $\beta \in [0, 1]$.

Under the assumptions of regularity and aperiodicity, the following proposition provides a temperature value suitable for the production of (α, β)-acceptable solutions, given any upper bound on e_1.

Proposition 1 *Let $e_1 \leq z \leq e_P$, solutions drawn from the stationary distribution at temperature*

$$T_f(z) = \frac{\beta(e_P - z)}{\log N - \log(1-\alpha)} \tag{5}$$

are (α, β)-acceptable.

Proof Let $\xi = e_1 + \beta(e_P - e_1)$ then

$$P(c(\omega) > \xi; T) = \sum_{i:e_i>\xi}^{P} \frac{N(i)e^{-\frac{e_i}{T}}}{K(T)} \leq \frac{e^{-\frac{\xi}{T}}}{K(T)} \underbrace{\sum_{i:e_i>\xi} N(i)}_{\leq N}$$

$$\leq N e^{-\frac{\beta e_P}{T}} e^{\frac{\beta e_1}{T}} \underbrace{\frac{e^{-\frac{e_1}{T}}}{K(T)}}_{\leq 1} \leq N e^{-\frac{\beta(e_P - z)}{T}}.$$

Where $N(i)$ is the number of solutions with value e_i and where

$$K(T) = \sum_{j=1}^{N} e^{-\frac{c(\omega_j)}{T}}.$$

Letting $N e^{-\frac{\beta(e_P-z)}{T}} = 1 - \alpha$ leads to Eq. 5. □

[4] Differential approximation theory is based on the notion of differential approximation ratio which measures how far the value of a solution is from the worst possible value. Its theoretical properties are investigated in Demange and Paschos (1996) (see also Monnot et al. 2003).

Let z_k denote the value of the best solution encountered up to iteration k, then the algorithm may be stopped as soon as $T \leq T_f(z_k)$. In other words, the above proposition allows to use the decreasing sequence of values of the best-so-far solution so as to generate an increasing sequence of *final temperature* values, given by Eq. 5, and to stop the algorithm as soon as it reaches the highest of these temperatures, that is *the better the best-so-far solution, the earlier the termination*.

In general, the choice of α and β leads the algorithm into the realm of relatively small temperature values. Hence, the ability for the simulated annealing algorithm to produce (α, β)-acceptable solutions depends on how well it is able to simulate the stationary distribution at such temperature values. In order to do so, the idea consists in starting the algorithm at a relatively high temperature, say T_0, where convergence towards the stationary distribution is extremely fast, and to decrease the temperature in such a way that the stationary distributions for two succeeding values are close to each other. In particular (Aarts and van Laarhoven 1985), choosing

$$T_{k+1} = \frac{T_k}{1 + \frac{\log(1+\delta)}{e_P+1} T_k} \tag{6}$$

guarantees that

$$|\pi_i^{(\infty)}(T_k) - \pi_i^{(\infty)}(T_{k+1})| \leq \delta,$$

where δ is a small positive real number. As a consequence, it is reasonable to expect that after decreasing the temperature only a few iterations are required in order to approach the new stationary distribution. Needless to emphasize that, despite of its reasonableness, this argument is of heuristic nature.

Note that the Markovian model of Sect. 2.1 has inspired many other cooling schedules. See Triki et al. (2005) for a recent survey.

3 Application to the PMP problem

3.1 Approximation measure

Provided that this paper is devoted to the study of approximate solution algorithms for the PMP problem, an approximation measure is required. Let c denote the value of the solution obtained using such an algorithm, the following approximation ratio shall be used to assess its quality

$$\beta = \frac{c - c^\star}{\sum_{m \in M} c_m - c^\star},$$

where $c^\star$ is the value of an optimal solution and where $\sum_{m \in M} c_m$ is the value of the worst possible solution which consists in interrupting all the moves.

This measure is quite natural as far as the PMP problem is concerned. On one hand, $1 - \beta$ can be interpreted as a *differential approximation ratio* (see Sect. 2.2). On

the other hand, it can also be seen as a *conventional approximation ratio* (Garey and Johnson 1979) for the maximization problem complementary to the PMP problem which asks to maximize the sum of the costs of the moves which are *not* interrupted.

3.2 Statement of the algorithm

As far as the PMP problem is concerned, Ω is the set of the $|M|!$ permutations of the moves and two such permutations are neighbours if one can be obtained from the other by exchanging the positions of only two moves and vice versa. It is obvious that the directed graph induced by such a neighbourhood function is strongly connected since any desired permutation can be obtained by starting with all elements in lexicographic order and then exchanging appropriate pairs of elements.

This, along with the theoretical aspects covered in the previous sections, leads to Algorithm 2. Its parameters are α, β and δ as well as the chain length (i.e., the number of iterations for each value of the temperature) which has been fixed to $|M|$ (a pragmatic as well as quite conventional choice). Also recall that for the PMP problem, the worst possible solution consists in interrupting all the moves, hence $e_P = \sum_{m \in M} c_m$.[5]

Algorithm 2 Simulated annealing applied to the PMP problem

Do $T \leftarrow \sum_{m \in M} c_m$.
Initialise π to an arbitrary permutation of the moves.
Do $c \leftarrow c(\pi)$, $\pi^\star \leftarrow \pi$ and $c^\star \leftarrow c$.
While $T \geq \frac{\beta(\sum_{m \in M} c_m - c^\star)}{\log|M|! - \log(1-\alpha)}$ do:
  For $k = 1$ to $|M|$ do:
    Choose m uniformly in M and m' uniformly in $M \setminus \{m\}$.
    Exchange $\pi(m)$ and $\pi(m')$ and do $c' \leftarrow c(\pi)$.
    Choose u uniformly in $[0, 1]$.
    If $u \leq e^{-\frac{c'-c}{T}}$ then[a]
      Do $c \leftarrow c'$.
      If $c < c^\star$ then do $\pi^\star \leftarrow \pi$ as well as $c^\star \leftarrow c$.
    Else
      Exchange $\pi(m)$ and $\pi(m')$.
    End.
  End.
  $T \leftarrow \frac{T}{1 + \frac{\log(1+\delta)}{1+\sum_{m \in M} c_m} T}$.
End.

[a] Again, note that $e^{-\frac{c(\omega')-c(\omega)}{T}} \geq 1$ when $c(\omega') \leq c(\omega)$

[5] Note that the calculation of the worst value is not always as straightforward as here and may even be as hard as finding the optimum value. See the discussion in Demange and Paschos (1996).

3.3 Building admissible solutions

Algorithm 2 requires a mechanism able to associate an admissible solution, hence a value, to any permutation $\pi : M \longrightarrow \{1, \ldots, |M|\}$.

First, let us remark that using the naive algorithm which tries to perform the moves in the order induced by π, interrupting those which are not feasible (see Algorithm 3), may lead to miss all optimum solutions. Figure 2 provides an example of an instance for which this happens. Indeed, the unique optimal solution clearly consists in interrupting move f and in then performing the other moves in the order $eadcb$. So either π orders move f before the other moves in which case Algorithm 3 performs it (since it is feasible) and later has to interrupt another move or π first orders a move other than f and since no such move is initially feasible the algorithm interrupts it. Both cases lead only to solution with value greater than 1, hence non optimal.

Algorithm 3 A naive algorithm which builds an admissible process move program from a permutation $\pi : M \longrightarrow \{1, \ldots, |M|\}$ of the moves. At the end of the algorithm the set I contains the interrupted moves ($c(\pi) = \sum_{m \in I} c_m$), then $m \notin I$ is performed before $m' \notin I$ if $\pi(m) < \pi(m')$

Do $I \leftarrow \emptyset$ and $L_u \leftarrow K_u$ for all $u \in U$.
For $i = 1$ to $|M| - 1$ do:
  $m \leftarrow \pi^{-1}(i)$.
  If $L_{t_m} \geq w_m$ then
    $L_{t_m} \leftarrow L_{t_m} - w_m$.
    $L_{s_m} \leftarrow L_{s_m} + w_m$.
  Else
    $L_{s_m} \leftarrow L_{s_m} + w_m$.
    $I \leftarrow I \cup \{m\}$.
  End.
End.

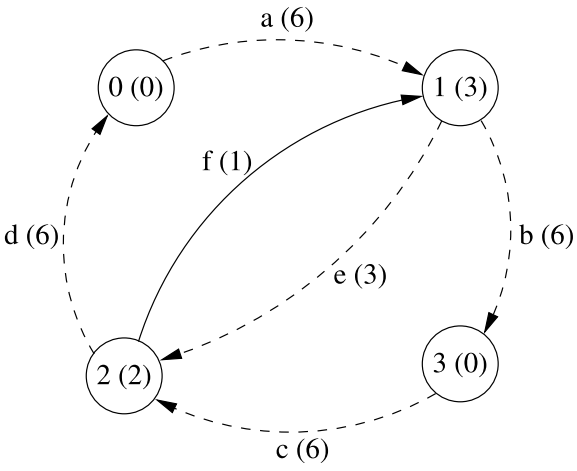

Fig. 2 An instance for which using Algorithm 3 leads to miss the unique optimal solution. The number in parenthesis indicates either the initial capacity of a processor (e.g. $K_1 = 3$) or the weight of a move (e.g., $w_a = 6$)

Algorithm 4 Construction of an admissible process move program from a permutation $\pi : M \to \{1, \ldots, |M|\}$ of the moves. At the end of the algorithm the set I contains the interrupted moves ($c(\pi) = \sum_{m \in I} c_m$), then $m \notin I$ is performed before $m' \notin I$ if $\pi(m) < \pi(m')$

Do $I \leftarrow M$.
Do $L_u \leftarrow K_u + \sum_{m \in S(u)} w_m$ and $\ell_u = L_u$ for all $u \in U$.
For $i = 1$ to $|M|$ do:
  $m \leftarrow \pi^{-1}(i)$.
  If $\ell_{s_m} \geq w_m$ then
    $\ell_{s_m} \leftarrow \ell_{s_m} - w_m$.
    $L_{t_m} \leftarrow L_{t_m} - w_m$.
    $\ell_{t_m} \leftarrow \min(\ell_{t_m}, L_{t_m})$.
    $I \leftarrow I \setminus \{m\}$.
  End.
End.

Algorithm 4 avoids this pitfall by starting from the worst solution, that is the solution which interrupts all the moves, and by trying to avoid these interruptions in the order specified by π. The principle of the algorithm is as follows. Initially all of the moves are interrupted. At the end of the kth step of the loop, $I^{(k)}$ (the content of variable I at this point of the execution) contains all the moves in $\{m \in M : \pi(m) > k\}$ as well as potentially some of the moves not in this set, then $m \notin I^{(k)}$ is performed before $m' \notin I^{(k)}$ if $\pi(m) < \pi(m')$. Let $m = \pi^{-1}(k+1)$, the interruption of m can then be avoided (i.e., m can be removed from I) as long as no capacity constraint is violated if the process associated to m remains on s_m during the execution of the solution obtained at the end of the kth step of the loop that is if

$$w_m \leq \min_{i=1,\ldots,k} \left(K_u + \sum_{m' \in S(u) \cap I^{(k)}} w_{m'} + \sum_{\substack{m' \in S(u) \setminus I^{(k)} \\ \pi(m') \leq i}} w_{m'} - \sum_{\substack{m' \in C(u) \setminus I^{(k)} \\ \pi(m') \leq i}} w_{m'} \right) = \ell_{s_m}^{(k)}, \tag{7}$$

where $\ell_{s_m}^{(k)}$ denotes the content of variable ℓ_{s_m} at the end of the kth step of the loop i.e., the residual capacity which is always available on s_m during the first k steps of the loop.

Proposition 2 implies that m is then feasible.

Proposition 2 *Let (I, σ) denote an admissible process move program, if the process associated to a move $m \in I$ can remain on s_m during the entire execution of (I, σ) then, after its execution, t_m has enough remaining capacity to host the process associated to m i.e., m is possible.*

Proof After performing all the moves in $M \setminus I$ the remaining capacity on t_m is equal to

$$K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} - \sum_{m' \in C(t_m) \setminus I} w_{m'}$$

Table 1 Trace of Algorithm 4 when executed on permutation $eadcbf$ of the moves of the instance shown in Fig. 2. At step 6, move f is added to I since $0 = \ell_1 < w_f = 1$

U	0	1	2	3	U	0	1	2	3	m
$\ell^{(0)}$	6	12	9	6	$L^{(0)}$	6	12	9	6	
$\ell^{(1)}$	6	9	6	6	$L^{(1)}$	6	12	6	6	e
$\ell^{(2)}$	0	6	6	6	$L^{(2)}$	6	6	6	6	a
$\ell^{(3)}$	0	6	0	6	$L^{(3)}$	0	6	6	6	d
$\ell^{(4)}$	0	6	0	0	$L^{(4)}$	0	6	0	6	c
$\ell^{(5)}$	0	0	0	0	$L^{(5)}$	0	6	0	0	b
$\ell^{(6)}$	0	0	0	0	$L^{(6)}$	0	6	0	0	f

and, from Eq. 3, we have

$$K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} - \sum_{m' \in C(t_m) \setminus I} w_{m'} \geq \sum_{m' \in C(t_m) \cap I} w_{m'} \geq w_m .$$

□

Table 1 illustrates that Algorithm 4 is indeed able to build the unique optimal solution to the instance shown in Fig. 2.

Lastly, it should be emphasized that Algorithm 4 can be extended to the multiple resource case in a straightforward manner.

3.4 Complexity

The following proposition quantifies the number of plateaux of temperature met by Algorithm 2.

Proposition 3 *Algorithm 2 meets* $O(\frac{|M| \log |M| \sum_{m \in M} c_m}{\log(1+\delta)})$ *plateaux of temperature.*

Proof Let $\gamma = \frac{\log(1+\delta)}{1+\sum_{m \in M} c_m}$, it is easy to see that the algorithm cooling schedule (Eq. 6) is such that

$$T_{k+l} = \frac{T_k}{1 + l\gamma T_k}.$$

Since at worst the algorithm remains stuck with a solution which interrupts all but one move and since such a solution has value at most $\sum_{m \in M} c_m - 1$, the number of plateaux, say Λ, is the solution of

$$\frac{T_0}{1 + \Lambda \gamma T_0} = \frac{\beta}{\log |M|! - \log(1-\alpha)},$$

that is

$$\begin{aligned} \Lambda &= \frac{(1 + \sum_{m \in M} c_m)(\log |M|! - \log(1-\alpha))}{\beta \log(1+\delta)} - \frac{1}{\gamma \sum_{m \in M} c_m} \\ &\propto \frac{\log |M|! \sum_{m \in M} c_m}{\log(1+\delta)} \\ &\approx \frac{(|M|-1) \log |M| \sum_{m \in M} c_m}{\log(1+\delta)}. \end{aligned}$$

□

Since Algorithm 4 clearly runs in $O(|M|)$ and provided that $|M|$ iterations are performed for each value of the temperature, Algorithm 2 runs in

$$O\left(\frac{|M|^3 \log|M| \sum_{m\in M} c_m}{\log(1+\delta)}\right).$$

Algorithm 2 is therefore pseudopolynomial.

Despite of this result, it should be emphasized that the method is relatively computationally expensive as, for example, the constant hidden in the O-notation has an order of magnitude of about 200 when $\beta = 0.05$ and $\delta = 0.1$. Of course, it is reasonable to expect that the worst case behavior occurs rarely in practice, as the algorithm usually finds reasonably good solutions fairly quickly (recall the discussion in Sect. 2.2).

4 Computational experiments

In this section, we report on computational experiments carried out so as to assess the practical ability of our simulated annealing algorithm *to produce* 5%*-acceptable solutions* ($\beta = 0.05$) *around 95 times out of 100* ($\alpha = 0.95$), which reflects "reasonable" quality expectations. Also, δ was set to 0.1. 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.

4.1 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 $\{1, \ldots, W\}$ 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 for which the 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 the introduction.

It should be emphasized 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 (Sirdey et al. 2003) the maximum *theoretical* load of a processor ranges (nonlinearly) from at most 50% (for a system with 2 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 simulated annealing algorithm was able to solve virtually all practical instances to optimality 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. As an example, on a set of 10 real instances of maximum practical size (72.3 moves, on average, for a system with 14 processors), the average time to optimality was 15.98 seconds.

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.

4.2 Computational results

In order to reasonably explore the (practically relevant part of the) problem space we have used the scheme of Sect. 4.1 to generate a set of 10 instances for each $|U| \in \{2, \ldots, 14\}$,[6] each $W \in \{10, 20, \ldots, 90, 100\}$ and $C = 100$. Hence a total of 1300 instances, amongst which only 1020 were considered of nontrivial 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. (2005a), the advantage being that the value of an optimum solution is known for many of these instances.

Just running the algorithm and hoping for the best is not entirely satisfactory for two main reasons:

1. It does not give any idea as to whether or not a 5%-acceptable solution has effectively been obtained, even if we have good reasons to believe it is often so.
2. It may induce unjustified computation time as, quite often, a 5%-acceptable solution is obtained fairly early.

In order to address the above issues we proceed in two steps:

1. A lower bound, denoted by l, is obtained by solving a linear programming relaxation of the problem using a cutting plane algorithm[7] (the details of which being out of the scope of this paper, see Sirdey and Kerivin 2007).

[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 2 and at most 14 processors (Sirdey et al. 2003).

[7] Although this relaxation involves exponentially many inequalities it can *theoretically* be solved in pseudopolynomial time. This follows from the pseudopolynomiality of the separation problem for these inequalities and from the well-known equivalence between optimization and separation (Schrijver 1986).

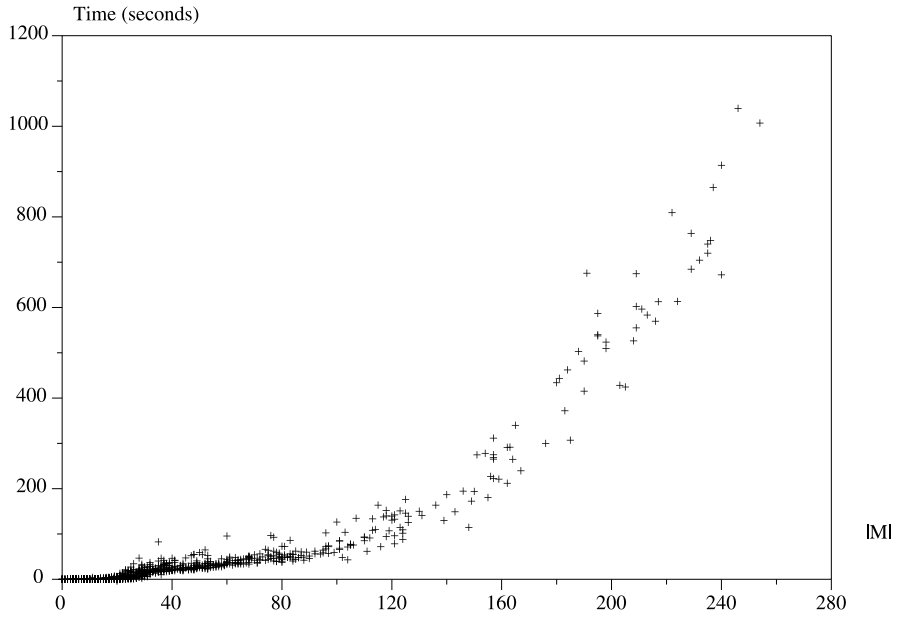

Fig. 3 Execution time of the algorithm in terms of the instance size

2. The simulated annealing algorithm is started and stopped as soon as a solution of value less than or equal to $0.95l + 0.05 \sum_m w_m$ is encountered, if this happens.

Figure 3 provides the execution time of the algorithm for each of the instances in the base.

On the overall instance base, the algorithm was able to conclude that a 5%-acceptable solution was obtained for 868 instances (85.01%). A large part of the 152 instances for which the algorithm was not able to reach this conclusion were located in the small number of moves and small number of processors area, where the LP bound seems to be of lesser quality. This is illustrated in Fig. 4. Moreover, the algorithm effectively obtained a 5%-acceptable solution for all but 23 of these 152 instances but was unable to prove it (we could reach this conclusion because we know the value of an optimal solution for many of these 152 instances). For all but 3 of these remaining 23 instances, all that was required to obtain a 5%-acceptable solution was to restart the simulated annealing step 1.7 times, on average. Finally, 10 trials of the simulated annealing step were not enough to provably obtain a 5%-acceptable solution in the case of only 3 instances, the characteristics of which being given in Table 2. Since the value of an optimal solution is unknown for these 3 instances we are unable to conclude, although it should be emphasized that the best solutions obtained are nearly 5%-acceptable.

Figure 4 provides the repartition of the 152 instances for which the algorithm was not able to conclude that a 5%-acceptable solution was obtained. A "+" indicates an instance for which the algorithm effectively obtained a 5%-acceptable solution but was unable to prove it, a "×" indicates an instance for which more than one run of the simulated annealing step were required to obtain a 5%-acceptable solution and

Table 2 Characteristics of the 3 instances for which 5%-acceptable solutions were not provably obtained, even after 10 trials of the simulated annealing step

$\|U\|$	$\|M\|$	OPT	Best β
13	36	?	5.65%
12	37	?	5.32%
10	24	?	5.90%

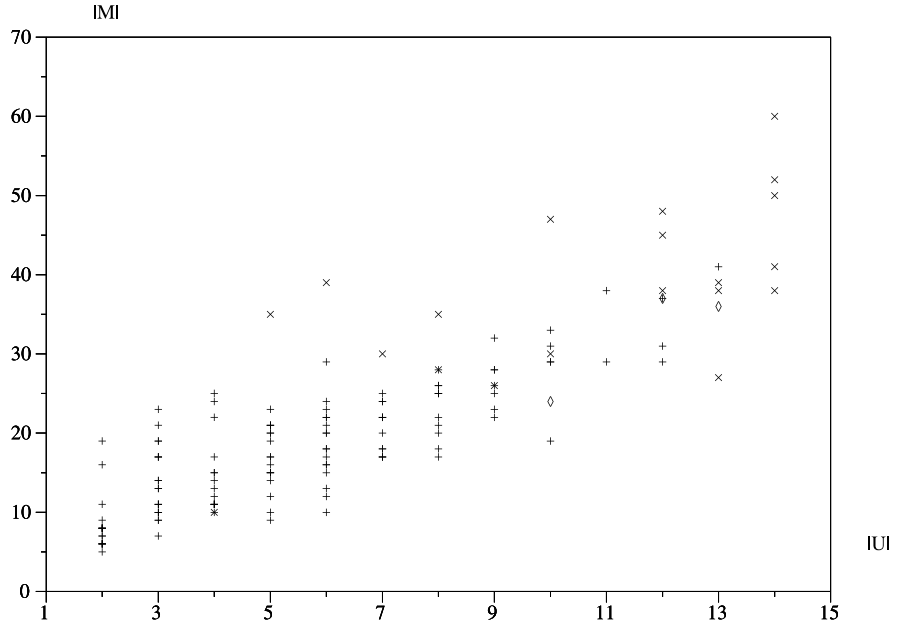

Fig. 4 Repartition of the 152 instances for which the algorithm was not able to conclude that a 5%-acceptable solution was obtained

a "◊" indicates an instance for which no 5%-acceptable solution has provably been obtained, even after 10 trials of the simulated annealing step.

Additionally, Table 3 illustrates the approximation performances of the algorithm by providing the percentage of instances solved within a given distance to optimality *in reality* (e.g. 1.76% of the instances were solved within a distance of 5 to 6% to optimality) and, to be fair, from the viewpoint of a "blind" user who has no knowledge on the value of an optimal solution other than provided by the lower bound. As already emphasized, the majority of the problematic instances (for the "blind" user) are located within the small number of moves (say less than 20) and small number of processors (say less than 7) area and are attributable to the relative weakness of the lower bound in this area.

If we ignore the above 3 problematic instances and coarsely estimate the probability for the simulated annealing scheme to effectively reach a 5%-acceptable solution (ratio of the number of successes over the number of trials) we obtain 0.9676, in remarkable agreement with the real value of α. Lastly, taking the pessimistic viewpoint and putting the 3 problematic instances back into the picture, that is considering 30 more trials and no additional success, leads to 0.9408, recall however that the ques-

Table 3 Percentage of instances solved within a given distance to optimality *in reality* and from the viewpoint of a "blind" user

	$\leq 5\%$	$]5\%, 6\%]$	$]6\%, 7\%]$	$]7\%, 8\%]$	$]8\%, 9\%]$	$]9\%, 10\%]$	$]10\%, 15\%]$	$]15\%, 18\%]$
Real	97.74%	1.76%	0.49%	0%	0%	0%	0%	0%
Blind	85.01%	4.61%	3.63%	2.06%	2.25%	0.88%	1.27%	0.20%

tion as to whether or not a 5%-acceptable solution has effectively been obtained is left opened for these 3 instances.

5 Conclusion

In this paper, we have proposed a simulated annealing-based approximation algorithm for the Process Move Programming 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).

We have introduced the notion of (α, β)-acceptable solution, where β is a measure of distance to optimality and α is the probability that it is achieved, and used the Markovian theory underlying the homogeneous simulated annealing algorithm to derive conditions under which such solutions may be produced. These results have then been used to design a pseudopolynomial time approximation algorithm for the PMP problem.

Although, at the end of the day, our reasoning is heuristic, since there is no theoretical guarantee that sufficient proximity to the stationary distribution is achieved at end of each plateau of temperature, we have performed extensive computational experiments which suggest that the algorithm meets its design intent.

In these experiments, the algorithm was set up to provide $(95\%, 5\%)$-acceptable solutions. Building on previous research on exact solution algorithms (Sirdey et al. 2005a), we have been able to demonstrate that the algorithm effectively obtained a 5%-acceptable solution for 97.74% of the 1020 instances (with up to 254 moves) on which it was tried. This, as well as the analysis of the number of retrials required to obtain a 5%-acceptable solution on the remaining instances, strongly suggests that the algorithm is indeed able to produce $(95\%, 5\%)$-acceptable solutions to the PMP problem.

Despite of the above, it should be emphasized that our simulated annealing algorithm is relatively computationally expensive. Faster, although having less sound theoretical foundations, algorithms are presently discussed in Sirdey et al. (2005).

References

Aarts, E.H.L., van Laarhoven, P.J.M.: Statistical cooling: a general approach to combinatorial optimization problems. Philips J. Res. **40**, 193–226 (1985)

Cerny, V.: Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. J. Optim. Theory Appl. **5**, 41–51 (1985)

Demange, M., Paschos, V.T.: On an approximation measure founded on the links between optimization and polynomial approximation theory. Theor. Comput. Sci. **158**, 117–141 (1996)

Garey, M.R., Johnson, D.S.: Computers and Intractability—A Guide to the Theory of *NP*-Completeness. Freeman, New York (1979)

Hajek, B., Sasaki, G.: Simulated annealing—to cool or not. Syst. Control Lett. **12**, 443–447 (1989)

Kemeny, J.G., Snell, J.L.: Finite Markov Chains. The University Series in Undergraduate Mathematics. D. van Nostrand Company/Princeton, New Jersey (1960)

Kirkpatrick, S., Gelatt, C.D. Jr.: Vecchi, M.P.: Optimization by simulated annealing. Science (May 1983)

Lundy, M., Mees, A.: Convergence of an annealing algorithm. Math. Program. **34**, 111–124 (1986)

Möbius, A., Neklioudov, A., Díaz-Sánchez, A., Hoffman, K.H., Fachat, A., Schreiber, M.: Optimization by thermal cycling. Phys. Rev. Lett. **79**, 4297–4301 (1997)

Monnot, J., Paschos, V.T., Toulouse, S.: Approximation Polynomiale des Problèmes *NP*-Difficiles. Optima Locaux et Rapport Différentiel. Hermès Science Publications, Lavoisier (2003)

Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York (1986)

Sirdey, R., Kerivin, H.: A branch-and-cut algorithm for a resource-constrained scheduling problem. PRAIRO Oper. Res. **41**, 235–251 (2007)

Sirdey, R., Plainfossé, D., Gauthier, J.-P.: 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 (2003)

Sirdey, R., Carlier, J., Nace, D.: A fast heuristic for a resource-constrained scheduling problem. Technical Report PE/BSC/INF/017254 V01/EN, Service d'architecture BSC, Nortel GSM Access R&D, France (2005)

Sirdey, R., Carlier, J., Kerivin, H., Nace, D.: On a resource-constrained scheduling problem with application to distributed systems reconfiguration. Eur. J. Oper. Res. **183**, 546–563 (2007)

Triki, E., Colette, Y., Siarry, P.: A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. Eur. J. Oper. Res. **166**, 77–92 (2005)

van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated Annealing: Theory and Applications. Mathematics and its Applications. Kluwer Academic, Dordrecht (1987)