A branch-and-cut algorithm for a resource-constrained scheduling problem

Renaud Sirdey, Hervé Kerivin

A branch-and-cut algorithm for a resource-constrained scheduling problem * †

Renaud Sirdey[a] [b] and Hervé Kerivin[c]

[a] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.
[b] UMR CNRS Heudiasyc (Université de Technologie de Compiègne), Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France.
[c] UMR CNRS Limos (Université de Clermont-Ferrand II), Complexe scientifique des Cézeaux, 63177 Aubière Cedex, France.

Corresponding author: R. Sirdey (*renauds@nortel.com*, +33 (0)1 69 55 41 18).

Submitted on March 16, 2006, 2006
Revised on May 23, 2006
Revised on July 18
Accepted on September 29, 2006

Abstract

This paper is devoted to the exact resolution of a strongly NP-hard resource-constrained scheduling problem, the *Process Move Programming problem*, which arises in relation to the operability of certain high-availability real-time distributed systems. Based on the study of the polytope defined as the convex hull of the incidence vectors of the admissible process move programs, we present a branch-and-cut algorithm along with extensive computational results demonstrating its practical relevance, in terms of both exact and approximate resolution when the instance size increases.

Keywords: Polyhedral combinatorics, scheduling, branch-and-cut, distributed systems, OR in telecommunications.

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

†Published in RAIRO—Operations Research **41**:235-251, 2007.

Introduction

In this paper, we present a branch-and-cut algorithm for the *Process Move Programming* (PMP) problem. This problem arises in relation to the operability of certain high-availability distributed switching systems. For example [27], 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 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.

We now proceed with a formal definition of the problem.

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 $f : P \longrightarrow U \cup \{u_\infty\}$, 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 $\bar{P}(f) = P(u_\infty; f)$ are not instantiated and, when this set is non empty, the system is in *degraded mode*.

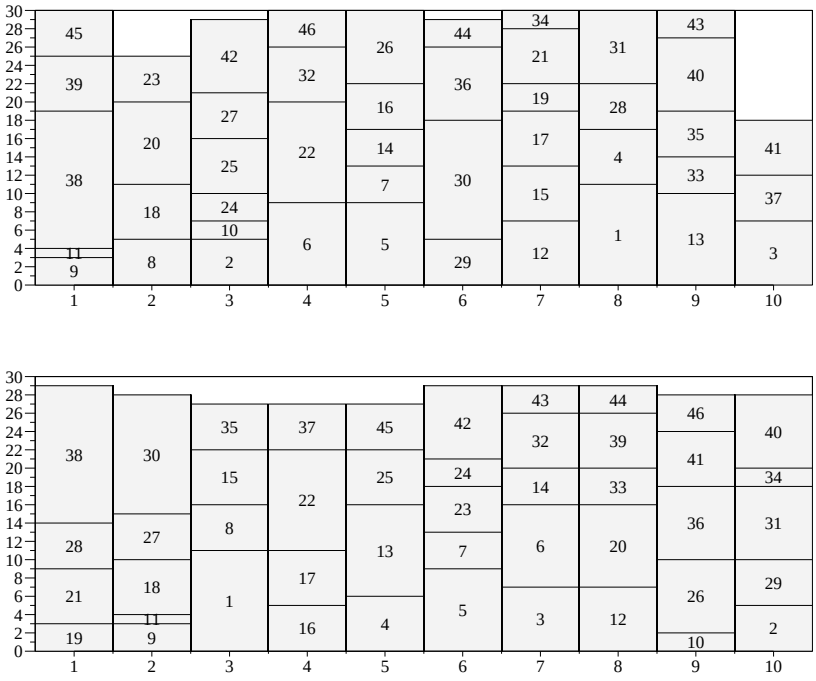

Figure 1: Example of an instance of the PMP problem.

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*[1].

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. [16]) 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 $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ is a bijection, defines an admissible *process move program*, 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 program if for all $m \in M \setminus I$ we have

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

[1]Throughout the rest of this paper, it is assumed that $\bar{P}(f_i) = \bar{P}(f_t) = \emptyset$. When this is not the case 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.

where $K_u = C_u - \sum_{p \in P(u;f_i)} w_p$ 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 $u \in U$

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

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.

Previous research on distributed system reconfiguration and other related problems [6, 7, 4, 15, 22, 3, 2] has mainly dealt with the approximate minimization of makespan related criteria under a set of constraints on the legal parallelism and has either ignored or only considered quite loose capacity constraints. As emphasized throughout this paper, the PMP problem or, more precisely, our model of the PMP problem shares some features with the linear ordering problem, which consists in finding a spanning acyclic tournament of maximum weight in a complete weighted digraph [21]. In terms of exact resolution, Gröstchel, Jünger and Reinelt [13] report solving real-world instances having up to 60 vertices with an algorithm that could be considered the first branch-and-cut ever. More recently, Christof and Reinelt [5] were able to solve "hard" instances having up to around 80 vertices with a parallel branch-and-cut algorithm using facets from low-dimensional polytopes. Lastly, Mitchell and Borchers [19] report solving instances with up to 250 vertices with a combined interior point/simplex cutting plane algorithm, though their instances appear easier than those of Christof and Reinelt. Needless to emphasize that, despite of certain structural similarities between the two problems, the presence of the capacity constraint implies that the PMP problem is in essence fairly different from the linear ordering problem.

Sirdey et al. [23] 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. Also, they provided an thorough literature survey. Additionally, approximate resolution algorithms have been proposed by Sirdey, Carlier and Nace [24, 25] (simulated annealing-based approach and Grasp-based approach, respectively). Lastly, in a series of two papers [26, 18] we have investigated the PMP prob-

lem from the polyhedral viewpoint: we formulated the PMP problem as an integer linear program and introduced several classes of valid inequalities, all of which being facet-defining for the associated polytope under mildly restrictive assumptions.

This paper intends to assess the practical relevance of the latter polyhedral results. To the best of the authors' knowledge, this is the first application of the polyhedral approach to a distributed system reconfiguration problem as well as the first attempt to tackle such a problem exactly, to the exception of the branch-and-bound algorithm presented in [23].

The paper is organized as follows. In Section 1, we provide the theoretical background laying at the basis of our algorithm. Our branch-and-cut algorithm is then presented in Section 2. Finally, we provide, in Section 3, extensive computational results which demonstrate the practical usefulness of the approach.

1 Polyhedral combinatorics of the PMP problem

In this section, we provide the theoretical background required in order to make this paper self-contained. We present an integer linear programming formulation of the problem along with a selection of facet-defining inequalities which are used in the algorithm. All the results in this section are proved in [26, 18], proofs are therefore omitted.

1.1 An integer linear programming formulation

In this section, we formulate the PMP problem as an integer linear program. We first focus on obtaining a formulation for the decision problem which asks whether or not there exists an admissible process move program of the form $(\emptyset, \sigma)$, that is an admissible program of zero cost. We subsequently refine the model so as to encompass the notion of interruption.

For each ordered pair of distinct moves of M, say m and m', we introduce the *linear ordering variables* [20]

$$\delta_{mm'} = \begin{cases} 1 & \text{if } m \text{ precedes } m', \\ 0 & \text{otherwise.} \end{cases}$$

In order for these variables to define a valid ordering, it is natural to ask for the following constraints to be satisfied

$$\begin{cases} \delta_{mm'} + \delta_{m'm} = 1 & \forall \{m, m'\} \subseteq M, \quad (4) \\ \delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M. \quad (5) \end{cases}$$

Constraints of type (4) simply express that either m precedes m' or m' precedes m. Constraints of type (5) are known as the *transitivity constraints* and simply state that if m precedes m' and if m' precedes m'' then m precedes m''. Along with the constraints

$$\delta_{mm'} \in \{0, 1\} \qquad m \neq m' \in M, \quad (6)$$

constraints of types (4) and (5) describe a *linear ordering polytope*, that is the convex hull of the incidence vectors of the linear orderings of the moves in M (see for example Grötschel, Jünger and Reinelt [12] or Fishburn [9] for details).

Since interruptions are (so far) disallowed, constraints of type (2) can be expressed as follows

$$w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} \delta_{m'm} - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} \qquad \forall m \in M. \quad (7)$$

It follows that any integral solution to the linear system of inequalities defined by the sets of constraints (4), (5), (6) and (7) (should such a solution exists) provides an admissible process move program of zero cost.

We now turn to the PMP problem and start by, for each move $m \in M$, introducing the variables

$$\delta_{mm} = \begin{cases} 1 & \text{if } m \text{ is interrupted,} \\ 0 & \text{otherwise.} \end{cases}$$

Constraints of type (2) can then be written as follows

$$(1 - \delta_{mm}) w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} (\delta_{m'm'} + \delta_{m'm}) - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm}, \quad (8)$$

for all $m \in M$. The transitivity constraints (5) remain unchanged and constraints of type (4) must be replaced by constraints

$$\delta_{mm'} + \delta_{m'm} = 1 - \max(\delta_{mm}, \delta_{m'm'}) \qquad \forall \{m, m'\} \subseteq M. \quad (9)$$

These constraints simply express that if either m or m' is interrupted then neither m precedes m' nor m' precedes m (recall that the interruptions are performed at the beginning). Additionally, constraints of type (9) are equivalent to the following set of constraints

$$\begin{cases} \delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall \{m, m'\} \subseteq M, \quad (10) \\ \delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M. \quad (11) \end{cases}$$

Inequalities of types (10) and (11) are respectively referred to as the 2-*clique* and 1-*unicycle* inequalities [26].

The resulting integer linear program for the process move programming problem is given in LP 2. The polytope associated to this program is hereafter referred to as the *process move program polytope* or, for short, the *PMP polytope* and denoted P^M_{PMP}.

Note that P^M_{PMP} is fully dimensional under mildly restrictive assumptions [18] and that full dimensionality is hereafter assumed unless stated otherwise.

1.2 Facets of the process move program polytope

It turns out that the 2-clique inequalities (10) and the 1-unicycle inequalities (11) along with inequalities $\delta_{mm'} \geq 0$ for all $m, m' \in M$ with $m \neq m'$ define facets of P^M_{PMP}.

This is not the case for inequalities $\delta_{mm} \geq 0$ for all $m \in M$, for inequalities $\delta_{mm'} \leq 1$ for all $m, m' \in M$ as well as for the transitivity constraints (5) and the capacity constraints (8).

The transitivity constraints can however be replaced by the *extended transitivity constraints*

$$\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} + \delta_{m'm'} \leq 1 \qquad m \neq m' \neq m'' \neq m \in M, \quad (12)$$

which are facet-defining for P^M_{PMP}.

Let $m_0 \in M$ and let $\emptyset \subset A \subseteq T(s_{m_0})$ and $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ be such that

$$\sum_{m \in A} w_m > K_{s_{m_0}} + \sum_{m \in \bar{B}} w_m, \quad (13)$$

with $\bar{B} = S(s_{m_0}) \setminus (B \cup \{m_0\})$ and define the *source cover inequality* generated by m_0, A and B as

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}). \quad (14)$$

$$
\begin{cases}
\text{Minimize } \displaystyle\sum_{m\in M} c_m\delta_{mm} & \\
\text{s. t.} & \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall\{m, m'\} \subseteq M, \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M, \\
\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M, \\
(1-\delta_{mm})w_m \leq K_{t_m} + \displaystyle\sum_{m'\in S(t_m)} w_{m'}(\delta_{m'm'} + \delta_{m'm}) - \sum_{m'\in T(t_m)\setminus\{m\}} w_{m'}\delta_{m'm} & \forall m \in M, \\
\delta_{mm'} \in \{0, 1\} & m, m' \in M.
\end{cases}
$$

Figure 2: Formulation of the PMP problem as an integer linear program.

Also, let $m_0 \in M$ and let $A \subseteq T(t_{m_0}) \setminus \{m_0\}$ and $\emptyset \subset B \subseteq S(t_{m_0})$ be such that

$$w_{m_0} + \sum_{m \in A} w_m > K_{t_{m_0}} + \sum_{m \in \bar{B}} w_m, \tag{15}$$

with $\bar{B} = S(t_{m_0}) \setminus B$, the *target cover inequality* generated by m_0, A and B is then defined as

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}). \tag{16}$$

These inequalities have the following meaning. Condition (13) (respectively (15)) expresses the fact that all of the moves in A (respectively $A \cup \{m_0\}$) cannot be performed if none of the moves in $B \cup \{m_0\}$ (respectively B) have been or, in other words, that performing all the moves in $\bar{B}$ does not free enough resources to perform all the moves in A (respectively $A \cup \{m_0\}$) and inequality (14) (respectively (16)) prevents that from happening as soon as m_0 is not interrupted.

It has been shown in [18] that the source and target cover inequalities are both valid (unconditionally) and facet-defining (under reasonably restrictive assumptions) for P^M_{PMP}. It also turns out, as we shall later see, that both the source and target cover inequalities can be separated in pseudopolynomial time.

Other classes of facet-defining inequalities for P^M_{PMP} which are not presently used in our branch-and-cut algorithm can be found in [26, 18], in particular, the 2-clique and 1-unicycle inequalities were generalized, yielding the k-clique ($k \geq 2$) and k-unicycle ($k \geq 1$) inequalities which are facet-defining for P^M_{PMP}.

2 A branch-and-cut algorithm

Our branch-and-cut algorithm is based on the linear relaxation which includes the trivial, 2-clique and 1-unicycle inequalities ($0 \leq \delta_{mm'} \leq 1$ for all $m, m' \in M$, (10) and (11), respectively) along with the extended transitivity and capacity constraints ((12) and (8), respectively) as well as the source and target cover inequalities ((14) and (16), respectively). Because all of these constraints are polynomial in number, apart from the latter two which can be separated in pseudopolynomial time, this linear relaxation can, in theory, be solved in pseudopolynomial time using the ellipsoid algorithm [14].

2.1 Solving the relaxation

In practice, the above linear relaxation is solved using a cutting-plane algorithm.

Separation-wise, the $O(|M|^2)$ 2-clique inequalities, the $O(|M|^2)$ 1-unicycle inequalities and the $O(|M|^3)$ extended transitivity inequalities are handled by brute-force.

The separation of the source and target cover inequalities, on the other hand, requires the resolution of $2|M|$ knapsack problems, each of these being solved in pseudopolynomial time using the well-known Bellman recursion [17].

Indeed, given $m_0 \in M$ and $\delta^\star \in \mathbb{R}^{|M|^2}$, the separation problem for the source cover inequalities asks for two sets $A \subseteq T(s_{m_0})$ and $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ which satisfy condition (13) and such that

$$\sum_{m \in A} \delta^\star_{mm_0} + \sum_{m \in B} \delta^\star_{m_0 m} > (|A| + |B| - 1)(1 - \delta^\star_{m_0 m_0}). \tag{17}$$

For $m \in T(s_{m_0}) \cup S(s_{m_0}) \setminus \{m_0\}$, let $x_m = 1$ if and only if either $m \in A$ or $m \in B$. Since $\sum_{m \in \bar{B}} w_m = \sum_{m \in S(s_{m_0})} w_m - \sum_{m \in B} w_m - w_{m_0}$, condition (13) can be rewritten

$$\sum_{m \in T(s_{m_0})} w_m x_m + \sum_{m \in S(s_{m_0}) \setminus \{m_0\}} w_m x_m \geq K_{s_{m_0}} + \sum_{m \in S(s_{m_0})} w_m - w_{m_0} + 1.$$

Since $|A| = \sum_{m \in T(s_{m_0})} x_m$ and $|B| = \sum_{m \in S(s_{m_0}) \setminus \{m_0\}} x_m$, inequality (17) can be rewritten (after rearrangement)

$$\sum_{m \in T(s_{m_0})} \xi_m x_m + \sum_{m \in S(s_{m_0}) \setminus \{m_0\}} \zeta_m x_m < 1 - \delta^\star_{m_0 m_0},$$

where $\xi_m = 1 - \delta^\star_{mm_0} - \delta^\star_{m_0 m_0}$ and $\zeta_m = 1 - \delta^\star_{m_0 m} - \delta^\star_{m_0 m_0}$. Letting $y_m = 1 - x_m$ leads to the following knapsack problem

$$\begin{cases} z = \text{Maximize} \displaystyle\sum_{m \in T(s_{m_0})} \xi_m y_m + \sum_{m \in S(s_{m_0}) \setminus \{m_0\}} \zeta_m y_m, \\ \text{s. t.} \\ \displaystyle\sum_{m \in T(s_{m_0})} w_m y_m + \sum_{m \in S(s_{m_0}) \setminus \{m_0\}} w_m y_m \leq \sum_{m \in T(s_{m_0})} w_m - K_{s_{m_0}} - 1, \\ y_m \in \{0, 1\},\ m \in T(s_{m_0}) \cup S(s_{m_0}) \setminus \{m_0\}. \end{cases}$$

Then, if z exists (that is the case only when $\sum_{m \in T(s_{m_0})} w_m - K_{s_{m_0}} - 1 \geq 0$) and if

$$z > \delta^{\star}_{m_0 m_0} - 1 + \sum_{m \in T(s_{m_0})} \xi_m + \sum_{m \in S(s_{m_0}) \setminus \{m_0\}} \zeta_m,$$

a violated source cover inequality generated by m_0 has been found. Otherwise, it can be concluded that none exists.

A similar argument applies to the separation problem for the target cover inequalities.

At the beginning of the cutting plane algorithm, only the trivial inequalities and the capacity constraints are included. Then, at each iteration, at most the 100 most violated 2-clique inequalities, at most the 100 most violated 1-unicycle inequalities and at most the 100 most violated extended transitivity constraints are added along with, for each move, the most violated source and target cover inequalities, if any.

The augmented linear relaxation is then reoptimized and, in order to keep its size reasonably small, we remove all the inequalities which have a positive slack at the current optimum to the exception of those initially present in the program, that is the trivial inequalities and the capacity constraints, although this might result in some inequalities being added and removed a few times. Note that this technique has already been used by Grötschel, Jünger and Reinelt on the linear ordering problem [11] which, as already stressed, shares some features (including having constraints which are polynomial in numbers and which cannot practically be handled in their entirety) with the present problem.

2.2 Overview of the algorithm

First, a "good" initial incumbent is obtained using the simulated annealing algorithm of Sirdey, Carlier and Nace [24], this algorithm being designed so as to produce (α, β)-acceptable solutions that is, given α and $\beta \in [0, 1]$, to produce, with probability at least α, solutions of value less than or equal to OPT + $\beta(S - \text{OPT})$, where OPT is the value of an optimal solution and $S = \sum_m c_m$ is the value of the worst possible one which consists in interrupting all the moves. Still, these are *not* theoretical guarantees but extensive computational experiments reported in [24] strongly suggest that, in practice, the algorithm actually meets its design intent when $\alpha = 0.95$ and $\beta = 0.05$ (these values also being those used in the present study).

At the root node of the search tree, the algorithm starts by solving the initial linear relaxation using the cutting-plane algorithm presented in Section 2.1. Early termination occurs if the ceiling of the current relaxation value becomes equal to the initial incumbent, in which case the optimality of the latter is established. Unless the solution of the relaxation is integral, hence optimal, branching occurs. Note that the ceiling of the value of the solution of the initial relaxation then serves as a global lower bound, hereafter denoted GLB.

Then, at each subsequent node of the search tree, the linear relaxation (in which some variables have been fixed) is solved, starting from the linear program obtained, before branching, at the parent node (note that the constraints are added or removed only locally). Early termination of the cutting-plane algorithm occurs either if the linear program is proven infeasible or if the ceiling of the current relaxation value becomes equal to the value of the incumbent. The incumbent is then updated if the solution of the relaxation is integral and, otherwise, branching occurs.

The search tree is traversed using depth-first search and a fairly simple branching scheme: a variable whose value, say v, is closest to $\frac{1}{2}$ in the relaxation is selected and 0 (respectively 1) is chosen first for branching if $v \leq \frac{1}{2}$ (respectively $v > \frac{1}{2}$), branching on 1 (respectively 0) subsequently occurs only if the incumbent is still greater than the global lower bound.

3 Computational experiments

In this section, we report on computational experiments carried out so as to assess the practical relevance of our branch-and-cut algorithm. These experiments have been performed on a Sun Ultra 10 workstation with a 440 MHz Sparc microprocessor, 512 MB of memory and the Solaris 5.8 operating system. The linear programs have been solved using COIN-OR implementation of the simplex algorithm [1]. Lastly, a time limit of four hours was imposed.

3.1 Instance generation

Given U the set of processors and C the processor capacity, an instance is generated as follows.

First, the set of candidate processes is built by drawing consumptions uniformly in $\{1, \ldots, C\}$ 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 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. Indeed, as far as the system to which this work is to be applied (see Sirdey, Plainfossé and Gauthier [27]) is concerned, the capacity constraints are fairly loose due to the fact that some spare capacity is provisioned for fault tolerance purpose and that 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. This preprocessing addresses the operational need to keep the number of moves as low as possible by solving a minimum cost bipartite matching problem so as to find the permutation of the processors which minimizes that number, this is achieved by letting $|P(u; f_i) \setminus P(u'; f_t)|$ be the cost of pairing processors u and u'.

Considering this, it turned out that our algorithm was able to solve virtually all practical instances to optimality within a few seconds and that, as a consequence, we had to consider more aggressive instance generation schemes, such as the above, in order to fairly evaluate its performances.

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.

3.2 Computational results

In our experiments, $|U|$, the number of processors, was ranging from 15 to 100 (with a step size of 5) and C, the processor capacity, was set to 100. For

each value of $|U|$ a set of 10 instances were generated. Hence, the algorithm was tried on 190 instances which sizes, in terms of number of moves, range from 17 to 184.

When the time limit was reached the algorithm output the best solution it found along with an upper bound on the optimality gap.

Given a solution of value z, distance to optimality was measured using the ratio

$$d(z) = \frac{z - \text{OPT}}{S - \text{OPT}},$$

where OPT (OPT, when unknown, being replaced by GLB) and $S = \sum_m c_m$ respectively denote the value of an optimal solution and of the worst possible one, which simply consists in interrupting all the moves. This is consistent with the definition of (α, β)-acceptability (section 2.2). Additionally, $1 - d(z)$ can be interpreted either as a *differential approximation ratio* (recall that differential approximation is concerned with how far the value of a solution is from the worst possible value [8]) or as a *conventional approximation ratio* [10] 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.

Table 1 illustrates the results obtained using our algorithm on a set of instances with around 45 moves (column "$|M|$") and 25 processors (column "$|U|$").Columns "z_0" and "$d(z_0)$" respectively indicate the value of the initial incumbent and the distance between that value and the optimum one. Columns "GLB", "d(GLB)", "# it." and "# cont." respectively provide the value of the initial relaxation, the distance between that value and the optimum one along with the number of iterations of the cutting-plane algorithm required to solve the relaxation and the number of constraints in the linear program before branching. Columns "$z^\star$", "# nodes" and "CPU" respectively indicate the value of an optimum solution, the number of nodes explored by the algorithm and the total running time. The average instance size, the average distance between the initial incumbent value and the optimal one as well as the average distance between the initial relaxation value and the optimal one are indicated at the bottom of the table.

Note that for the fourth instance an integral solution was obtained before branching and that for the sixth one the initial incumbent was proven optimal before completing the resolution of the initial relaxation.

Table 2 provides a summary of the results which were obtained using our branch-and-cut algorithm on the overall instance set. For each value of $|U|$, column "$\overline{|M|}$" provides the average number of moves of the instances in

$\|U\|$	$\|M\|$	z_0	$d(z_0)$	GLB	d(GLB)	# it.	# cont.	$z^\star$	# nodes	CPU
25	45	84	0.00%	78	0.33%	144	3667	84	5	113.88 s
25	38	173	3.40%	110	0.00%	53	2394	110	3	36.35 s
25	50	161	2.26%	110	0.24%	225	4352	115	13	330.35 s
25	37	218	3.51%	155	0.00%	48	2529	155	1	49.78 s
25	36	206	2.96%	153	0.00%	54	2263	153	4	31.28 s
25	41	68	0.00%	68	0.00%	$\geq$99	-	68	1	70.85 s
25	40	118	2.01%	77	0.22%	73	2657	81	15	52.79 s
25	55	74	1.23%	49	0.00%	325	4652	49	5	329.06 s
25	47	158	1.27%	135	0.00%	151	3700	135	6	130.71 s
25	42	128	3.30%	69	0.00%	43	2671	69	2	39.09 s
	43.1		1.99%		0.08%					

Table 1: Illustration of the results obtained using our algorithm on a set of 10 instances with $|U| = 25$.

the set, columns "# inst." and "# solved" respectively indicate the number of instances which were generated and the number of instances which the algorithm was able to solve to optimality within the four hours time limit, additionally, columns "$\bar{d}(z_0)$", "$\bar{d}$(GLB)", "$\bar{d}(z_f)$" give upper bounds on, respectively, the average distance between the initial incumbent value and the optimal one, the average distance between the initial relaxation value and the optimal one as well as the average optimality gap. For example, the third row ($|U| = 25$) is a summary of Table 1.

In terms of exact resolution, our algorithm was quite successful up to $|U| = 45$ (i.e., with instances of size up to around 80 moves) in the sense that most instances were either solved to optimality or with an integrality gap of less than 2% within the allowed four hours.

As the instance size increased, along with $|U|$, fewer instances ended up being solved to optimality, within the four hours limit. Nevertheless, the algorithm is still practically relevant as it was able to find solutions with an optimality gap of less than 5% for all but 7 of the instances on which it was tried.

Overall, the biggest instance which was solved to optimality within the four hours time limit had size 119 ($|U| = 70$), and was solved in 2 h 11 m 57 s, and the smallest instance which was not solved to optimality had size 49 ($|U| = 30$), though the optimality gap was less than or equal to 1.02%.

$\|U\|$	$\overline{\|M\|}$	# inst.	# solved	$\bar{d}(z_0)$	$\bar{d}$(GLB)	$\bar{d}(z_f)$
15	23.2	10	10	1.92%	0.07%	0.00%
20	34.1	10	10	1.20%	0.33%	0.00%
25	43.1	10	10	1.99%	0.08%	0.00%
30	53.1	10	6	≤2.01%	≤1.05%	≤0.74%
35	61.8	10	7	≤2.38%	≤0.98%	≤0.86%
40	67.5	10	6	≤2.23%	≤0.48%	≤0.43%
45	76.5	10	6	≤2.50%	≤1.15%	≤1.02%
50	88.5	10	3	≤2.69%	≤1.80%	≤1.72%
55	93.6	10	4	≤3.22%	≤1.34%	≤1.27%
60	108.4	10	2	≤2.80%	≤2.52%	≤2.52%
75	117.7	10	0	≤3.69%	≤3.69%	≤3.69%
70	122.3	10	1	≤3.32%	≤3.02%	≤3.00%
75	131.3	10	0	≤4.08%	≤4.08%	≤4.08%
80	137.4	10	0	≤3.30%	≤3.30%	≤3.30%
85	148.5	10	0	≤3.67%	≤3.67%	≤3.67%
90	157.4	10	0	≤3.79%	≤3.79%	≤3.79%
95	169.2	10	0	≤4.77%	≤4.77%	≤4.77%
100	177.2	10	0	≤4.62%	≤4.62%	≤4.62%

Table 2: Summary of the results obtained using our branch-and-cut algorithm on the overall instance set.

Additionally, for only one instance, of size 172, the bound on the optimality gap was greater than 6% (actually 6.12%). Table 3 indicates for each range of the optimality gap, the size of the smallest instance which was not solved and the size of the biggest instance which was solved with a gap bound in that range.

Gap	$]0\%, 1\%]$	$]1\%, 2\%]$	$]2\%, 3\%]$	$]3\%, 4\%]$	$]4\%, 5\%]$	$]5\%, 6\%]$
Smallest	49	56	56	85	119	172
Biggest	101	139	155	180	184	184

Table 3: Sizes of the smallest instance which was not solved and of the biggest instance which was solved within a given optimality gap bound range.

Lastly, it should be emphasized that from $|U| = 75$ onward, the initial incumbent was rarely improved during the branch-and-cut phase. This latter phase, however, allowed to obtain a reasonable estimate of the optimality gap. This illustrates the relevance of hybridizing a carefully designed metaheuristic, such as the simulated annealing algorithm used to obtain the initial incumbent [24], and a polyhedral bound when tackling problems in the realm of bigger instances.

Empirically, the branch-and-cut algorithm presented in this paper appears complementary to the combinatorial algorithm presented in [23]. Although the latter algorithm turns out being faster when dealing with small instances, due to the comparatively low per node computational cost, as well as with instances having relatively homogeneous process weights, mainly due to the presence of dominance relations which are particularly efficient in that context, it suffers from the lack of a strong lower bound. As emphasized by the above results, our branch-and-cut algorithm does not suffer from such a drawback: the strength of the linear programming bound presented in this paper allows it to tackle, either exactly or within a few percents to optimality, instances which are out of the reach of the aforementioned combinatorial algorithm. Table 4 illustrates this on the set of 10 instances of Table 1[2]. Indeed, although the branch-and-bound algorithm was able to find an optimum solution for 6 of the 10 instances, it was able to complete the optimality proof for only 3 of them, within a one hour time limit. Moreover, on these 3 instances, the calculation time was longer than that of the branch-and-cut

[2]To be fair, the branch-and-bound algorithm was also provided with the initial incumbent obtained with the simulated annealing algorithm discussed in Section 2.2.

algorithm.

$\|M\|$	45	38	50	37	36
BC	113.88 s	36.35 s	330.35 s	49.78 s	31.28 s
BB	$> 3600^*$ s	103.08 s	> 3600 s	> 3600 s	2779.19 s
$\|M\|$	41	40	55	47	42
BC	70.85 s	52.79 s	329.06 s	130.71 s	39.09 s
BB	$> 3600^*$ s	381.07 s	> 3600 s	> 3600 s	$> 3600^*$ s

Table 4: Experimental comparison between our branch-and-cut algorithm and the branch-and-bound algorithm presented in [23] on the set of 10 instances of Table 1. A "$*$" indicates that the algorithm was able to find an optimum solution but not to complete the optimality proof.

4 Conclusion

In this paper, we have proposed a branch-and-cut 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 by Sirdey, Plainfossé and Gauthier [27].

The main ingredient of our branch-and-cut algorithm is a linear relaxation which is made up of exponentially many inequalities which are facet-defining for the PMP polytope. This relaxation, which can theoretically be solved in pseudopolynomial time, is solved, at each node of the search tree, using a cutting-plane algorithm.

From an industrial perspective, it can be considered that the PMP problem is solved by this algorithm as it is able to close virtually all practical instances within a few seconds. Additionally, we have reported on computational experiments illustrating the practical relevance of the algorithm when used to solve instances significantly harder than those occurring in practice, in terms both of size and tightness of the capacity constraints. Indeed the

algorithm was able to solve instances with up to 119 moves (70 processors) within a four hours time limit. Although one should only expect to have good chances to solve instances of size up to around 80 moves within a four hours limit, our experiments still suggest that, when the instance size increases, the truncated version of the algorithm has fairly good approximate resolution capabilities as it was able to provably obtain solutions with an optimality gap of less than 5% for most instances of size up to around 180 moves, still within the four hours time limit.

Lastly, note that further research work will be carried out so as to assess the practical relevance of the classes of facet-defining inequalities identified in [18] and which, for simplicity sake, are so far not used in our branch-and-cut algorithm.

Acknowledgements

The authors wish to thank the two anonymous referees for several suggestions that led to improvements in the paper.

References

[1] Computational infrastructure for operations research (`www.coin-or.org`), last access on July the 18th, 2006.

[2] G. Aggarwal, R. Motwani, and A. Zhu. The load rebalancing problem. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 258–265, 2003.

[3] E. Anderson, J. Hall, J. Hartline, M. Hobbs, A. R. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. An experimental study of data migration algorithms. In *Proceedings of the 5th International Workshop on Algorithm Engineering*, Lecture Notes in Computer Science, page 145. Springer, 2001.

[4] J. Carlier. Le problème de l'ordonnancement des paiements de dettes. *RAIRO—Operations Research*, 18, février 1984.

[5] T. Christof and G. Reinelt. Algorithmic aspects of using small instance relaxations in parallel branch-and-cut. Technical report, Université de Heidelberg, 1998.

[6] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. Lapaugh. Scheduling file transfers in distributed networks. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 254–266, 1983.

[7] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. Lapaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14, 1985.

[8] M. Demange and V. T. Paschos. On an approximation measure founded on the links between optimization and polynomial approximation theory. *Theoretical Computer Science*, 158:117–141, 1996.

[9] P. C. Fishburn. Induced binary probabilities and the linear ordering polytope: a status report. *Mathematical Social Sciences*, 23:67–80, 1992.

[10] M. R. Garey and D. S. Johnson. *Computers and intractability—A guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

[11] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[12] M. Grötschel, M. Jünger, and G. Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33:43–60, 1985.

[13] M. Grötschel, M. Jünger, and G. Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33:28–42, 1985.

[14] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[15] J. Hall, J. Hartline, A. R. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 620–629, 2001.

[16] P. Jalote. *Fault tolerance in distributed systems*. Distributed Systems. Prentice Hall, 1994.

[17] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[18] H. Kerivin and R. Sirdey. Polyhedral combinatorics of a resource-constrained ordering problem part II: on the process move program polytope. Technical Report PE/BSC/INF/017913 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, France, 2006.

[19] J. E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In *High performance optimization*, pages 349–366. Kluwer, 2000.

[20] M. Queyranne and A. S. Schulz. Polyhedral approaches to machine scheduling. Technical Report 408/1994, Université de Technologie de Berlin, 1994.

[21] G. Reinelt. *The linear ordering problem: algorithms and applications*, volume 8 of *Research and exposition in mathematics*. Heldermann Verlag Berlin, 1985.

[22] J. C. Saia. Data migration with edge capacities and machine speeds. Technical report, Université de Washington, 2001.

[23] R. Sirdey, J. Carlier, H. Kerivin, and D. Nace. On a resource-constrained scheduling problem with application to distributed systems reconfiguration. *European Journal of Operational Research*, 183:546–563, 2007.

[24] R. Sirdey, J. Carlier, and D. Nace. Approximate resolution of a resource-constrained scheduling problem. *Journal of Heuristics (in press)*, 2005.

[25] R. Sirdey, J. Carlier, and D. Nace. 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.

[26] R. Sirdey and H. Kerivin. Polyhedral combinatorics of a resource-constrained ordering problem part I: on the partial linear ordering polytope. Technical Report PE/BSC/INF/017912 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, France, 2006.

[27] R. Sirdey, D. Plainfossé, and J.-P. Gauthier. 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*, pages 532–539, 2003.