CIRRELT
Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport
Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

An Iterated Local Search Heuristic for Multi-Capacity Bin Packing and Machine Reassignment Problems

Renaud Masson
Thibaut Vidal
Julien Michallet
Puca Huachi Vaz Penna
Vinicius Petrucci
Anand Subramanian
Hugues Dubedout

November 2012

CIRRELT-2012-70

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

An Iterated Local Search Heuristic for Multi-Capacity Bin Packing and Machine Reassignment Problems

Renaud Masson[1,2], Thibaut Vidal[1,3,*], Julien Michallet[3], Puca Huachi Vaz Penna[4], Vinicius Petrucci[4], Anand Subramanian[5], Hugues Dubedout[6]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Mathematics and Industrial Engineering, École Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, Canada H3C 3A7

[3] Laboratoire d'optimisation des systèmes industriels (LOSI), Université de Technologie de Troyes, 12, rue Marie Curie, B.P. 2060, 10010 Troyes, Cedex, France

[4] Universidade Federal Fluminense, Instituto de Computação, Rua Passo da Pátria, 156 Bloco E, 3º andar, São Domingos Niterói-RJ, CEP: 24210-240, Brazil

[5] Universidade Federal da Paraíba, Departamento de Engenhaira de Produção, Via Expressa Padre Zé, s.n, Jardim Cidade Universitária, João Pessoa, PB, CEP 58051-001, Brazil

[6] Écoles des Mines de Nantes, Institut de Recherche en Communications et Cybernétique (IRCCyN), 1, rue de la Noë, BP 92101, 44321 Nantes Cedex 3, France

Abstract. This paper proposes an efficient Multi-Start Iterated Local Search (MS-ILS) metaheuristic for Multi-Capacity Bin Packing Problems (MCBPP) and Machine Reassignment Problems (MRP). The MCBPP is a generalization of the classical bin-packing problem in which the machine (bin) capacity and task (item) sizes are given by multiple (resource) dimensions. The MRP is a challenging and novel optimization problem, aimed at maximizing the usage of available machines by reallocating tasks/processes among those machines in a cost-efficient manner, while fulfilling several capacity, conflict, and dependency-related constraints. The proposed MS-ILS approach relies on simple neighborhoods as well as problem-tailored shaking procedures. We perform computational experiments on MRP benchmark instances containing between 100 and 50,000 processes. Near-optimum multi-resource allocation and scheduling solutions are obtained while meeting specified processing-time requirements (on the order of minutes). In particular, for 15/28 instances with more than 1,000 processes, the gap between the solution value and a lower bound measure are smaller than 0.1%. Our optimization method is also applied to solve classical benchmark instances for the MCBPP, yielding the best known solutions and optimum ones in most cases. In addition, several upper bounds for non-solved problems were improved.

Keywords: Metaheuristics, iterated local search, multi-capacity bin packing, machine reassignment.

Acknowledgements. Partial funding for this project has been provided by the Champagne-Ardenne regional council, France, the French European fund for Economic and Regional Development (FEDER), the Natural Sciences and Engineering Council of Canada (NSERC), the Fonds de recherche du Québec - Nature et technologies (FRQNT), the Brazilian Government agencies CNPQq and CAPES. This support is gratefully acknowledged.

---

* Corresponding author: Thibaut.Vidal@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2012

1 Introduction

Emerging cloud computing platforms, such as Google Apps and Amazon EC2, are an effective way to support many kinds of Internet-based services (Armbrust et al. 2010). These platforms are deployed in large-scale server clusters (data centers) composed of several hundreds or thousands of inter-connected machines and allow hosting multiple individual services characterized by a set of interacting processes/tasks. The machines in these platforms provide different types of resources, such as computing (CPU), memory (RAM) and storage (DISK). The processes of a given service are allowed to run on those machines and consume their resources.

The problem of allocating a set of processes across a pool of server machines is basically a bin-packing problem where each of the machines is a bin. The goal is to allocate as many processes as possible into each bin, while respecting multiple capacity constraints. This problem is referred in the literature as the Multi-Capacity Bin Packing Problem (MCBPP), also known as the Vector Bin Packing Problem. In a more complex scenario, we consider the Machine Reassignment Problem (MRP) which consists of optimizing the usage of available machine resources by reallocating processes to different machines in a cost-efficient manner. An optimal reassignment decision must maximize load balancing/spreading objectives while satisfying resource capacity constraints with minimal migration costs of processes among the machines. This difficult optimization problem was originally proposed by Google for the 2012 ROADEF/EURO challenge (Roadef 2012).

Despite its relevance in today's data center applications, the specific characteristics of the MRP have not been adequately and extensively addressed in the literature. In fact, few resolution approaches are capable of handling very large size instances, such as those proposed by Google, that consist of several constraints related to service dependencies, spread and conflicts in order to meet load balancing and fault tolerance needs.

Typically, the workload of multiple services in a cluster can change over time. This requires to periodically solve an optimization problem and use the solution to reconfigure the cluster. We refer to Petrucci et al. (2011) for an optimization control loop design and implementation in a real server cluster, which can benefit from the underlying optimization problem investigated and solved in this work.

In this paper, we propose a Multi-Start Iterated Local Search (MS-ILS) tailored for large-scale machine reassignment and packing problems with multiple resources. Our method iteratively applies a local-search procedure to improve the solutions, along with shaking moves to escape from local optima. To reduce the computational time, shaking moves/restarts are dynamically triggered whenever the improvement is estimated to be too small with respect to the total solution cost. Extensive experimental evaluations demonstrate the remarkable performance of the proposed method on MRP instances with up to 50,000 processes. The MCBPP is also addressed by iteratively decrementing the number of machines and solving a Lagrangian relaxation of the packing problem. The method performs very well on classical instances for this problem, provided by Caprara and Toth (2001). All the upper bounds and optimal solutions found by previous algorithms are retrieved, and several of them are improved.

The remainder of this paper is organized as follows. Section 2 formally defines the MCBPP and presents related works. The MRP definition and a mathematical model are given in Section 3, along with a simple approach for computing lower bounds used to assess the quality of the solutions produced by our method. Section 4 describes the proposed MS-ILS metaheuristic. Section 5 is dedicated to the computational experiments, describing the benchmark instances

used, parameters calibration and sensitivity analysis, and finally the results of MS-ILS on the MRP and MCBPP. Conclusions and future directions are discussed in Section 6.

2 The Multi-Capacity Bin Packing Problem

The MCBPP can be formally defined as follows. Let P be a set of objects and R a set of resources. Any object $p \in P$ is characterized by a vector of $|R|$ resource consumptions. A set of homogeneous bins is available to pack the objects. Let C_r be the bin capacity for a resource $r \in R$. The objective of the MCBPP is to pack the objects of P in as few bins as possible, without exceeding the capacity of the bins for any resource.

The literature on bin packing is wide, as well as the number of related applications. Two main families of multi-dimensional bin packing problems shall be discerned: those concerned with "geometrical packing" of objects in space, which are outside the scope of the present paper, and those concerned with "vector packing", where each object is simply characterized by some independent resource consumptions without any immediate notion of geometrical space.

A number of related papers specific to vector packing in the context of multiprocessor systems were proposed over the past 40 years. Some early heuristics were first introduced by Graham (1972) and Kou and Markowsky (1977). Concerning the MCBPP with two dimensions, Spieksma (1994) developed a branch-and-bound algorithm, while Caprara and Toth (2001) put forward exact and heuristic approaches as well as worst-case performance analysis. Kellerer and Kotov (2003) developed a $\mathcal{O}(n \log n)$ approximation algorithm with absolute worst-case performance ratio 2. Chang et al. (2005) modeled the real-life problem of packing steel products into special containers as a two dimensional MCBPP and proposed a heuristic algorithm.

For problems with more resources, a heuristic approach using set-covering formulation was proposed by Monaci and Toth (2006). Wilcox et al. (2011) developed a genetic algorithm to solve the MCBPP as a task placement problem for packing processes onto a cluster of servers. Panigrahy et al. (2011) studied heuristics for task placement problems inspired by FFD (first-fit decreasing) variants. Shachnai and Tamir (2012) derived approximation algorithms for data placement problem in the context of multimedia storage systems. The multicomputer task allocation as MCBPP was modeled by Beck (1996), and a comparison between 256 packing algorithms was conducted. Leinberger and Karypis (1999) addressed the MCBPP for application in parallel computing system. Finally, Epstein and Tassa (2003) proposed a general framework for MCBPP and studied the monotonicity of the objective function through problems that occurs in video transmission.

Surveying this literature, it is notable that very few advanced metaheuristics are currently available for the MCBPP. Instances of moderate size (less than 200 objects) have been most often addressed. We contribute to fill this methodological gap by proposing a simple and efficient metaheuristic which can efficiently deal with large-scale MCBPP and MRP instances with up to 50,000 objects.

3 The Machine Reassignment Problem

The Machine Reassignment Problem seeks to optimize the assignment of a set of *processes*, P, intended to run on a set M of heterogeneous *machines*. A set R of *resources* is considered, the *capacity* of a machine $m \in M$ for any resource $r \in R$ is noted $C_{m,r}$. Any process $p \in P$ *requires* $R_{p,r}$ units of each resource $r \in R$. The MRP assumes a feasible initial solution S_{init}, in which

each process $p \in P$ is assigned to a machine $M_0(p)$. This initial assignment, however, may be improved by moving some processes among machines while respecting several constraints. No time dimension is considered in the model and all processes are moved simultaneously without any time-processing delay. The MRP includes several constraints (Section 3.1), and objectives combined in a weighted sum function (Section 3.2). Given those constraints and objectives, we propose a mathematical formulation for the MRP (Section 3.3) and lower bound (LB) measure (Section 3.4).

3.1 Constraints

• A **capacity constraint** limits, for each resource r, the total resource consumption of the processes running on a machine m to $C_{m,r}$. Note that a subset $TR \subseteq R$ of resources is considered as *transient*. During the migration process from one machine to the other, such transient resources are consumed on both current and newly allocated machines.
• Let S be a set of *services*. Any process p belongs to a unique service. A **conflict constraint** then forbids two processes of the same service from being run on the same machine.
• Machines are grouped into *locations* and *neighborhoods* denoted by L and N, respectively. Each machine is associated to only one location and neighborhood. Each service $s \in S$ must respect a **spread constraint** that imposes the minimum number of distinct locations, given by SM_s, on which the processes of the service should run.
• Additionally, one service can be dependent upon other services. The set of services on which service $s \in S$ depends is denoted by D^s. The **dependency constraint** requires that if a given process of service s runs on a machine of neighborhood $n \in N$, at least one process of each service of D^s has to run on a machine that belongs to neighborhood n.

3.2 Objectives

• For each machine m and each resource r a safety capacity, $SC_{m,r}$, is defined. Then, the **load cost** is defined as the sum of excess resource consumptions, over $SC_{m,r}$, on all machines and resources.
• Another cost, called **balance cost**, penalizes unbalanced free resources within machines. Indeed, having a large quantity of a free resource $r_1 \in R$ on a machine, but no more free resource of another type, often results in wasted capacity. Let $FR(m, r)$ be the amount of free resource $r \in R$ on machine $m \in M$ w.r.t. to its capacity $C_{m,r}$. The balance cost is decomposed into a set B of cost components. Each such component $b \in B$ is associated with a cost $C_{\text{BAL}}(b) = \sum_{m \in M} \max(0, T^b FR(m, r_1^b) - FR(m, r_2^b))$, where r_1^b and r_2^b are two resources, and T^b is a ratio.
• Any movement of a process p leads to a **process move cost** Φ_p. This move cost depends upon the process. Any process moved between a machine pair (m_1, m_2) leads to an additional **machine move cost** $\Phi_{m_1 m_2}$. Finally, **service move cost** proportional to the maximum number of moved processes from the same service is incurred.

3.3 Integer Programming Formulation

We introduce a mathematical formulation for the MRP in Expressions (1-12), which may be solved to optimality for small/medium instances using standard solvers. This formulation relies on the binary decision variables $x_{m,p}$, taking value one if and only if the process $p \in P$ is

operated on machine $m \in M$. Also, the binary variables $y_{l,s}$ take value one if and only if at least one process of service $s \in S$ is operated in location $l \in L$, and the binary variables $z_{n,p}$ indicate if process $p \in P$ is in neighborhood $n \in N$.

$$\begin{aligned}
&\text{Minimize} \sum_{r \in R} \Phi_r \sum_{m \in M} \max \left\{ 0, (\sum_{p \in P} x_{p,m} R_{p,r}) - SC_{m,r} \right\} \\
&+ \sum_{b \in B} \Phi_b \sum_{m \in M} \max \left\{ 0, T^b (C_{m,r_1^b} - \sum_{p \in P} x_{p,m} R_{p,r_1^b}) - (C_{m,r_2^b} - \sum_{p \in P} x_{p,m} R_{p,r_2^b}) \right\} \\
&+ \sum_{p \in P} \sum_{m \in M \setminus M_0(p)} (\Phi_p + \Phi_{M_0(p),m}) x_{m,p} + \Phi_s \max_{s \in S} (\sum_{p \in s} \sum_{m \in M \setminus M_0(p)} x_{m,p}) \qquad (1)
\end{aligned}$$

$$\text{s.t.} \quad \sum_{m \in M} x_{m,p} = 1 \qquad \forall p \in P \qquad (2)$$

$$\sum_{p \in P} x_{m,p} R_{p,r} \leq C_{m,r} \qquad \forall m \in M, r \in R \setminus TR \qquad (3)$$

$$\sum_{p \in P, M_0(p) \neq m} x_{m,p} R_{p,r} + \sum_{p \in P, M_0(p) = m} R_{p,r} \leq C_{m,r} \qquad \forall r \in TR, m \in M \qquad (4)$$

$$\sum_{p \in s} x_{m,p} \leq 1 \qquad \forall m \in M, s \in S \qquad (5)$$

$$\sum_{p \in s} \sum_{m \in l} x_{m,p} \geq y_{l,s} \qquad \forall l \in L, s \in S \qquad (6)$$

$$\sum_{l \in L} y_{l,s} \geq SM_s \qquad \forall s \in S \qquad (7)$$

$$z_{n,p} = \sum_{m \in n} x_{m,p} \qquad \forall n \in N, p \in P \qquad (8)$$

$$z_{n,p} \leq \sum_{k \in s_b} z_{n,k} \qquad \forall n \in N, p \in P, s_b \in D^{s(p)} \qquad (9)$$

$$x_{m,p} \in \{0,1\} \qquad \forall m \in M, p \in P \qquad (10)$$

$$y_{l,s} \in \{0,1\} \qquad \forall l \in L, s \in S \qquad (11)$$

$$z_{n,p} \in \{0,1\} \qquad \forall n \in N, p \in P \qquad (12)$$

The objective function (1) aims at minimizing the weighted sum of the five MRP costs, presented in the same order as previously (Section 3.2). Following the description from Section 3.1, constraints (2) guarantee that each process is assigned to one machine. Constraints (3) state that the capacity of each machine for each resource must be respected. Constraints (4) enforce transient resources capacity constraints. Constraints (5) prevent two processes of the same service to be assigned to the same machine. Constraints (6) and (7) enforce the spread constraints. Constraints (8) and (9) enforce the dependency constraints.

3.4 A simple lower bound for the MRP

For the largest instances considered in this work (50,000 processes and 5,000 machines), even computing the optimal value of the linear relaxation of Problem (1-12) is not practical. Thus, we introduce a simpler Lower Bound (LB) for the MRP which can be evaluated in linear time relatively to the problem size. This LB is an essential tool to assess the effectiveness of the proposed method. Let us first introduce Problem (P'):

$$\text{(P')} \quad \text{Minimize} \sum_{r\in R} \Phi_r \sum_{m\in M} \max\left\{0, (\sum_{p\in P} x_{p,m}R_{p,r}) - SC_{m,r}\right\}$$
$$+ \sum_{b\in B} \Phi_b \sum_{m\in M} \max\left\{0, T^b(C_{m,r_1^b} - \sum_{p\in P} x_{p,m}R_{p,r_1^b}) - (C_{m,r_2^b} - \sum_{p\in P} x_{p,m}R_{p,r_2^b})\right\} \tag{13}$$

$$\text{s.t.} \quad \sum_{m\in M} x_{m,p} = 1 \qquad \forall p \in P \tag{14}$$

$$x_{m,p} \in \{0,1\} \qquad \forall m \in M, p \in P \tag{15}$$

It can be observed that (P') is a relaxation of the MRP in which the objectives considered are only the load cost and the balance cost. The conflict, dependency, capacity and spread constraints are relaxed. We also put forward the following relations:

$$\max\left\{0, \sum_{p\in P} R_{p,r} - \sum_{m\in M} SC_{m,r}\right\} = \max\left\{0, \sum_{m\in M}(\sum_{p\in P} x_{p,m}R_{p,r}) - SC_{m,r}\right\}$$
$$\leq \sum_{m\in M} \max\left\{0, (\sum_{p\in P} x_{p,m}R_{p,r}) - SC_{m,r}\right\} \quad \forall r \in R \tag{16}$$

$$\max\left\{0, T^b(\sum_{m\in M} C_{m,r_{b_1}} - \sum_{p\in P} R_{p,r_{b_1}}) - (\sum_{m\in M} C_{m,r_{b_2}} - \sum_{p\in P} R_{p,r_{b_2}})\right\}$$
$$\leq \sum_{m\in M} \max\left\{0, T^b(C_{m,r_1^b} - \sum_{p\in P} x_{p,m}R_{p,r_1^b}) - (C_{m,r_2^b} - \sum_{p\in P} x_{p,m}R_{p,r_2^b})\right\} \quad \forall b \in B \tag{17}$$

Relations (16) and (17) lead to the following LB for (P'). As (P') is a relaxation of the MRP, the expression above is also a LB of the MRP. The LB measure is given as follows:

$$LB = \max\left\{0, \sum_{p\in P} R_{p,r} - \sum_{m\in M} SC_{m,r}\right\}$$
$$+ \max\left\{0, T^b(\sum_{m\in M} C_{m,r_{b_1}} - \sum_{p\in P} R_{p,r_{b_1}}) - (\sum_{m\in M} C_{m,r_{b_2}} - \sum_{p\in P} R_{p,r_{b_2}})\right\} \tag{18}$$

4 Proposed MS-ILS Metaheuristic

This section describes the proposed Multi-Start Iterated Local Search (MS-ILS) for machine reassignment and multi-capacity bin packing problems. The general description of the method is given in Sections 4.1 to 4.3, and some specificities related to its application to the MCBPP are discussed in Section 4.4.

4.1 General algorithm structure

The MS-ILS metaheuristic combines a Local-Search (LS) improvement procedure with problem-tailored shaking moves and specialized restart procedures. The MRP assumes an initial feasible solution S_{INIT}, which can be directly used as starting point by the ILS. The stochastic nature of the method enables to explore different parts of the search space when applying restarts.

The pseudo-code of MS-ILS is presented in Algorithm 1. In this algorithm, as well as in the remainder of this paper, we assume the availability of a pseudo-random number generator Alea() which produces uniformly distributed random real numbers in $[0, 1]$. As described in lines 6-7, the local search is applied with equal probability on either the best solution, with a view of intensifying the search around good solution characteristics, or on the current solution to give more search opportunities. The resulting solution replaces the best known in case of improvement, and a shaking move is applied. Two types of alternating shaking moves are used. LS and shaking are applied as long as the improvement of the current best solution is significant, a *Restart* being triggered whenever the best solution improvement has been smaller than a threshold γ_{RESTART} during the last five LS and shaking iterations. The method terminates once a maximum time limit T_{max} is reached, and returns the best overall solution.

Algorithm 1 Multi-start iterated local search for the MRP

1: $S_{\text{BESTALL}} \leftarrow S_{\text{INIT}}$
2: **while** time() $< T_{max}$ **do**
3: $S_{\text{CURR}} \leftarrow S_{\text{INIT}}$; $S_{\text{BEST}} \leftarrow S_{\text{INIT}}$ *//New start from the initial solution*
4: $It_{\text{SHAK}} = 0$ *//Counter for the number of shaking phases*
5: **while** the relative improvement of the best solution during the last 5 shaking phases is greater than $\gamma_{\text{RESTART}}\%$ **do**
6: **if** $(Alea() < 0.5)$ *//50% chance to work on the current best solution*
7: **then** $S_{\text{CURR}} \leftarrow \text{LocalSearch}(S_{\text{BEST}})$
8: **else** $S_{\text{CURR}} \leftarrow \text{LocalSearch}(S_{\text{CURR}})$
9: **if** $\text{Cost}(S_{\text{CURR}}) < \text{Cost}(S_{\text{BEST}})$ **then** $S_{\text{BEST}} \leftarrow S_{\text{CURR}}$
10: **if** $(It_{\text{SHAK}}\%2 == 0)$ *//Two alternating shaking moves*
11: **then** $S_{\text{CURR}} \leftarrow \text{Shaking1}(S_{\text{CURR}})$
12: **else** $S_{\text{CURR}} \leftarrow \text{Shaking2}(S_{\text{CURR}})$
13: **if** $\text{Cost}(S_{\text{BEST}}) < \text{Cost}(S_{\text{BESTALL}})$ **then** $S_{\text{BESTALL}} \leftarrow S_{\text{BEST}}$
14: **return** S_{BESTALL} *//Returning best overall solution*

4.2 Local-Search Procedure

The LS procedure consists in iteratively exhausting a set of moves and applying only improving ones. During each iteration of the LS, a subset of machines $\overline{M}$ is selected and, for every machine

$m \in \overline{M}$, a subset of moves involving m is evaluated. The best move found on m is then applied. At each loop, the selection of machines in $\overline{M}$ is driven by their associated cost, e.g. the sum of the load and balance cost of processes on this machine. This selection is done by ranking the machines in decreasing order of cost, and then choosing the first $|M| \times [0.1 + 0.9 \times Alea()^2]$ machines in this order.

Algorithm 2 LS improvement procedure

```
1: while the relative solution improvement during the last 5 loops is greater than γ_SHAKING% do
2:    Sort(Machine List)
3:    fractionMachines ← |M| × [0.1 + 0.9 × Alea()^2]
4:    for all i ∈ {1,...,fractionMachines} do
5:       m ← Machine List[i]
6:       exploreRelocateMoves(m)
7:       exploreSwapMoves(m)
8:       S_CURR ← performBestMove(S_CURR)
9: return S_CURR
```

Two kinds of moves are considered in the LS. The first one is a relocate move that transfers a single process from machine m to another one. The second kind of move is a swap move that consists of interchanging one or two processes from machine m with one or two process from a distinct machine. After exploring the relocate and swap moves of a machine, the best improving move (if any) is performed. The general rationale of this LS is described in Algorithm 2. The LS continues as long as the total solution improvement during the last five loops is significant, i.e. larger than a threshold $\gamma_{\text{SHAKING}}\%$. A detailed description of each subset of moves is given below (illustrated in Figure 1).

Relocate neighborhood. The relocate neighborhood includes all moves that consist of relocating a single process from $m \in M$ to a different machine. This neighborhood contains $|P_m^{sol}|(|M|-1)$ moves, where P_m^{sol} stands for the set of processes currently assigned to machine m in solution sol. As the number of processes and machines considered in the instances can be large (up to 50,000 processes and 5,000 machines), exploring complete neighborhoods may be prohibitively slow. Therefore, only a random subset of this neighborhood is explored, in such a way that on average $\varphi/3$ moves are evaluated for a given machine, where φ is a parameter of the method.

Swap neighborhood. The swap neighborhood includes all moves that swap one or two processes from m with one or two processes from a different machine. Again, only a randomly-selected subset of φ moves from this neighborhood is explored. Note that the randomly-selected moves may be different from one LS iteration to another.

4.3 Shaking operators

Shaking operators are potentially-deteriorating moves which provide the means to escape from locally optimal solutions. Two shaking operators are considered in the proposed method, and called in turn. Note that only feasible shaking moves are applied.

The **Home Relocate** shaking operator selects randomly ξ processes that are currently not hosted on their initial machine, and relocates them to their initial machine. This move tends

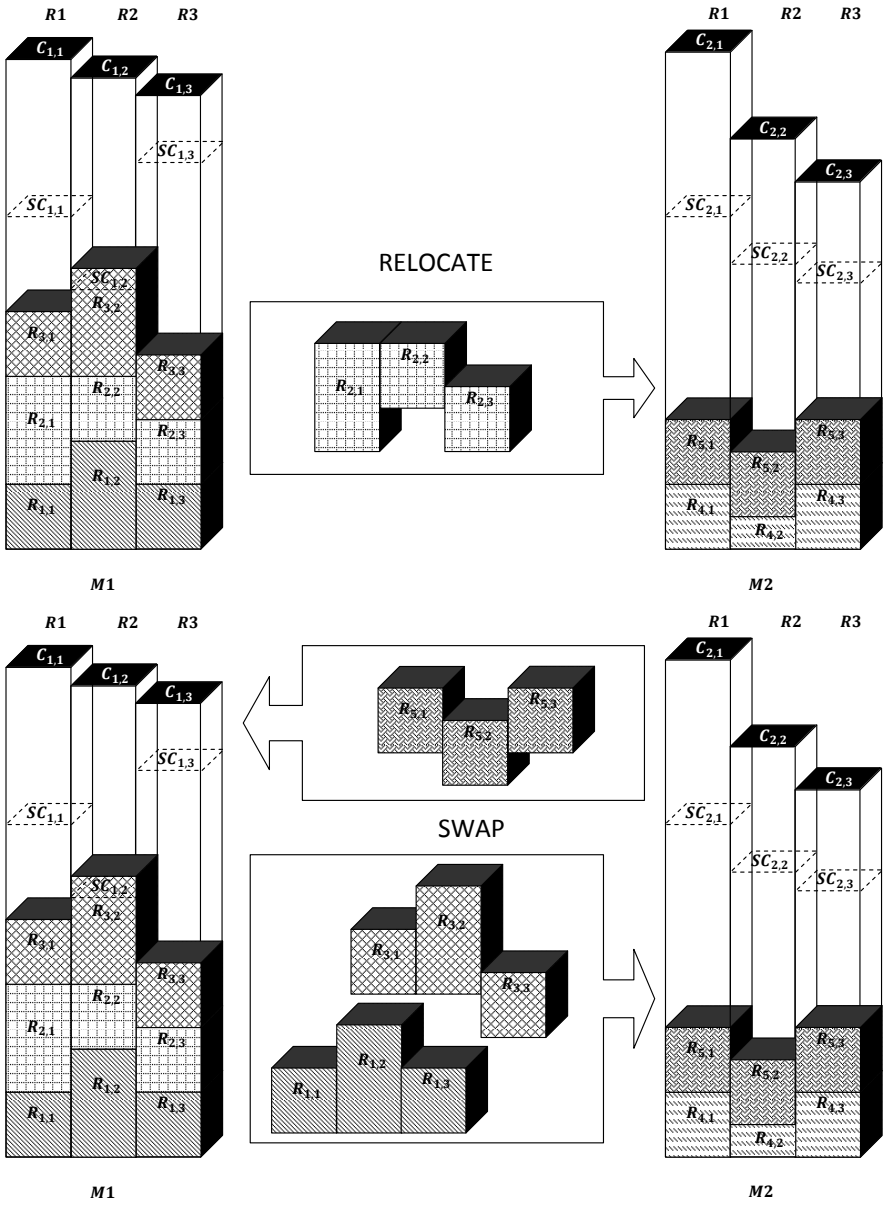


Figure 1: Illustration of the relocate and exchange/swap move explored by our algorithm.

to be particularly efficient for reducing the process move costs in MRP context.

The **K-Swap** shaking operator randomly selects ξ times a pair of machines, and performs a swap of random groups of either 3, 4, or 5 processes, with equal probability, among these machines. These two operators modify the solution in such a way that it cannot be directly reversed by simply applying a single local-search move.

4.4 Addressing the MCBPP

This concise description highlights the simplicity of the method, contrasting with the apparent complexity of the MRP. In addition, since the MRP is a fairly general problem, it encompasses many packing settings as special cases, with multiple resources, potential heterogeneous bins, spread/conflict constraints. We adapted our methodology to address the MCBPP, which is highly relevant in the context of multiprocessors systems, to demonstrate the method capability in producing high-quality solutions on other classic problem variants.

In the MCBPP context, it was necessary to tackle the objective of bin minimization rather than process re-arrangement. We thus applied a *relax-and-decrement* strategy, in which the algorithm iteratively imposes the number of machines (i.e. bins) and works on a relaxed version of the MCBPP with linearly-penalized capacity violations. This relaxed packing problem, for a fixed number of bins, can be easily modeled as a MRP with safety capacities and load costs. A feasible packing is found with the current number of bins whenever the penalty reaches zero. This number is iteratively decremented until no zero-penalty packing can be found, and the minimum number of bins yielding a feasible solution is returned.

5 Computational experiments

Extensive computational experiments have been conducted to calibrate the search parameters, evaluate their impact, and analyze the performance and scalability of the approach on MRP and MCBPP benchmark instances of various sizes. This section describes the benchmark instances used for the tests (Section 5.1), presents the parameter calibration and sensitivity analyses (Section 5.3), and finally assesses the performance of the method on the MRP and MCBPP benchmark instances (Sections 5.4-5.5) with different run-time limits. The MS-ILS algorithm has been coded in C++ and compiled using "g++ -O3". All tests were conducted on an Opteron 2.4 GHz with 4 GB of RAM memory running Linux OpenSuse 11.1.

5.1 Benchmark Instances

The benchmark dataset of the MRP considered in this work is composed of 3 sets (Sets A,B and X) of 10 instances that were used in the 2012 ROADEF Challenge (Roadef 2012). The size of these instances ranges from 4 machines and 100 processes to 5000 machines and 50000 processes. The instances of Set A tends to be of smaller size than the others. Sets B and X are larger, and very similar in nature.

The MCBPP instances used to evaluate the performance of our heuristic are taken from Spieksma (1994), Caprara and Toth (2001), and also addressed in Monaci and Toth (2006). A total of 10 different classes of instances are presented. Each class is composed of 40 instances, containing 10 instances of four different sizes. Classes 2,3,4,5 and 8 are known to be easily solvable by simple greedy heuristics (Monaci and Toth 2006), and thus we focus our experiments on the remaining classes, leading to a total of 200 instances. We refer to Caprara and Toth (2001) for a thorough description of the specificities of each class. In particular, the instances of Class 10 have been generated by *cutting* the bin resources into triplets of objects, in such a way that there is not a single unit of capacity slack in these solutions. Thus, for this class, the optimal solutions are in most cases known as a consequence of the instance generation process, but not a result of any bin packing algorithm.

5.2 Solution-quality measure for the MRP

Some MRP instances considered in this work may have arbitrarily low optimal costs. Thus, assessing the solution quality as a Gap(%) from the best solution cost would give too much importance to problems for which the expected final solution value is low, and, in some cases even, dividing per zero. We thus rely on another alternative, which involves measuring and comparing the *cost gains* relative to the initial solution S_{INIT}.

Define the *cost gain* $G(X)$ of a method X yielding a final solution $S_{\text{FINAL}}(X)$ as $G(X) = S_{\text{INIT}} - S_{\text{FINAL}}(X)$. Also, define the upper bound on cost gains G_{MAX} as the difference between the initial solution value and the lower bound (LB) on the solution cost: $G_{\text{MAX}} = S_{\text{INIT}} - LB$. Finally define the *score* $C(X)$ of a method X, in Equation (19), as its relative percentage of gap to G_{MAX} in terms of cost gain (percentage taken relatively to the initial solution cost).

$$C(X) = 100 \times \frac{G_{\text{MAX}} - G(X)}{S_{\text{INIT}}} = 100 \times \frac{S_{\text{FINAL}}(X) - LB}{S_{\text{INIT}}} \tag{19}$$

A score of 1 means that it is not possible, in the better case, to achieve an additional system improvement by more than 1% of the initial cost. This performance indicator matches most industrial application realities.

5.3 Parameters Calibration

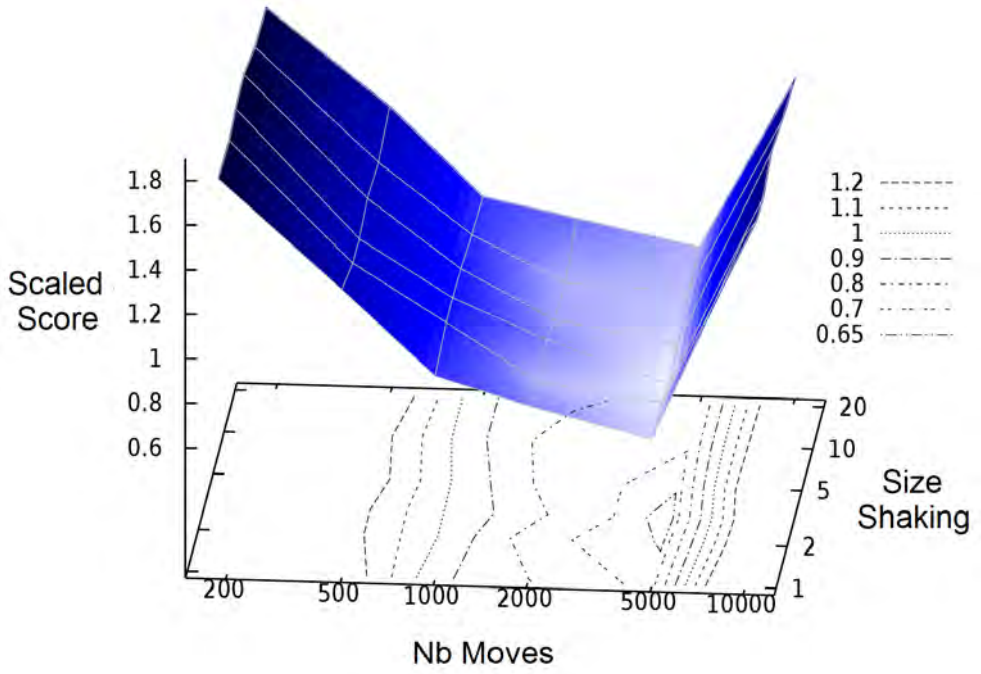
The MS-ILS algorithm relies on four main parameters: the thresholds γ_{RESTART} and γ_{SHAKING} for triggering a restart or a shaking phase. The size of the shaking operator ξ and the pruning criteria φ.

The first two parameters were independently calibrated in order to trigger the restart or the shaking when visibly no significant solution improvement occurs. Our experiments revealed that the best values for these parameters were $\gamma_{\text{RESTART}} = 0.5\%$ and $\gamma_{\text{SHAKING}} = 0.05\%$. These parameters have an influence on the run time by avoiding unnecessary search phases, but not a large impact on solution quality.

In contrast, the solution quality is very sensitive to the two last parameters, ξ and φ. We thus present the calibration results on 4 training instances: A2-2, B-1, B-5, and B-10 with different parameters values, selected in a range that appeared to be relevant during prior analyses: $\xi \in \{1, 2, 3, 5, 10, 20\}$ and $\varphi \in \{200, 500, 1000, 2000, 5000, 10000\}$. The instances of set X were not included in the calibration, since experiments on this set aims to reproduce a real setting where no prior instance-specific calibration has been conducted. Average results on 30 runs are considered. Also, to avoid giving higher importance to instances for which scores are larger (such as A2-2 for example), we applied an affine scaling of the score values in such a way that for each instance the best scaled score becomes 0 and the worst 1. Figure 2 reports the sum of the four scaled scores for each parameter setting.

It can be observed that the good parameter values are located around $\xi = 3$ and $\varphi = 5000$. The method seems very sensitive to the value of the pruning parameter φ. More precisely, high values for this parameter means evaluating more moves on a machine before applying the best, leading to more elitist but time-consuming searches. This kind of elitism appeared to be critical on some instances, especially B-1, where good decisions in the beginning of the search have a high impact on the overall performance. However, if the elitism is too high, e.g. with $\varphi = 10000$, the method does not have enough time to converge to a local optimum of the LS, and thus the solution quality rapidly decreases.

Figure 2: Average scaled score on four training instances with different parameter settings



$\varphi \setminus \xi$	1	2	**3**	5	10	20
200	1.780	1.763	**1.796**	1.811	1.745	1.732
500	1.300	1.230	**1.229**	1.272	1.245	1.307
1000	0.923	0.977	**0.976**	0.969	0.933	0.914
2000	0.803	0.720	**0.803**	0.752	0.727	0.828
5000	**0.669**	**0.645**	**0.616**	**0.658**	**0.707**	**0.716**
10000	1.594	1.470	**1.462**	1.528	1.484	1.463

5.4 MS-ILS performance on MRP benchmark instances

The MS-ILS metaheuristic was run 40 times for each instance. A time limit of five minutes is imposed in order to match practical application contexts. Table 1 presents the results found on the MRP benchmark dataset. In the columns are reported, in turn, the instance name, number of machines and processes, initial solution cost, our best LB, average solution and score of MS-ILS on the 40 runs, and best solution and score of MS-ILS. This best LB has been computed, for the Set A, by running the CPLEX optimization solver for solving the mathematical model of Section 3.3, and for Set B and X, using the expressions presented in Section 3.4. The last lines of the Table present the average score per group of instances.

In general, the quality of the solutions produced by the method is very high, and for 21/30 instances the score is smaller than 1%, meaning that more than 99% of the possible solution improvements have been achieved. On Set A, some results with larger gaps are reported. We believe that for these instances, the LB that we use may be of insufficient quality, thus complicating the task to assess the method performance on these few instances. The method seems to produce high-quality solutions in a consistent manner, with only a small gap between

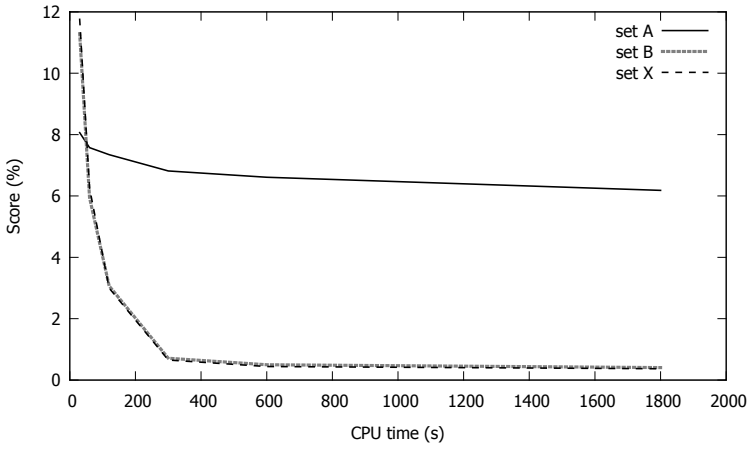
Inst	$\|M\|$	$\|P\|$	Init. Cost	LB	MS-ILS Avg		MS-ILS Best	
A1-1	4	100	49528750	44306501	44306501	0.00%	44306501	0.00%
A1-2	100	1000	1061649570	777531000	787593254	0.95%	780499081	0.28%
A1-3	100	1000	583662270	583005918	583006343	0.00%	583006015	0.00%
A1-4	50	100	632499600	242406000	263449642	3.33%	258024574	2.47%
A1-5	12	1000	782189690	727578000	727578639	0.00%	727578412	0.00%
A2-1	100	1000	391189190	126	515038	0.13%	167	0.00%
A2-2	100	1000	1876768120	537253000	1006392553	25.00%	970536821	23.09%
A2-3	100	1000	2272487840	1031400000	1488682473	20.12%	1452810819	18.54%
A2-4	50	1000	3223516130	1680230000	1721276657	1.27%	1695897404	0.49%
A2-5	50	1000	787355300	307403000	444758671	17.45%	412613505	13.36%
Score Set A						6.82 %		5.82%
B1-1	100	5000	7644173180	3290754940	3643207680	4.61%	3516215073	2.95%
B1-2	100	5000	5181493830	1015153860	1034641974	0.38%	1027393159	0.24%
B1-3	100	20000	6336834660	156631070	165037128	0.13%	158027548	0.02%
B1-4	1000	10000	9209576380	4677767120	4677962023	0.00%	4677940074	0.00%
B1-5	100	40000	12426813010	922858550	926221613	0.03%	923857499	0.01%
B1-6	200	40000	12749861240	9525841820	9525923099	0.00%	9525913044	0.00%
B1-7	4000	40000	37946901700	14833996360	15372961904	1.42%	15244960848	1.08%
B1-8	100	50000	14068207250	1214153440	1220521241	0.05%	1214930327	0.01%
B1-9	1000	50000	23234641520	15885369400	15885635887	0.00%	15885617841	0.00%
B1-10	5000	50000	42220868760	18048006980	18155878491	0.26%	18093202104	0.11%
Score Set B						0.69 %		0.44%
X1-1	100	5000	7422426760	3023565050	3348966927	4.38%	3209874890	2.51%
X1-2	100	5000	5103634830	1001403470	1026504981	0.49%	1018646825	0.34%
X1-3	100	20000	6119933380	0	3151834	0.05%	1965401	0.03%
X1-4	1000	10000	9207188610	4721558880	4721814589	0.00%	4721786173	0.00%
X1-5	100	40000	12369526590	0	696174	0.01%	615277	0.00%
X1-6	200	40000	12753566360	9546930520	9547005000	0.00%	9546992887	0.00%
X1-7	4000	40000	37763791230	14251967330	14893088510	1.70%	14701830252	1.19%
X1-8	100	50000	11611565600	0	369803	0.00%	309080	0.00%
X1-9	1000	50000	23146106380	16125494300	16125775950	0.00%	16125753242	0.00%
X1-10	5000	50000	42201640770	17815790830	17928266694	0.27%	17867789754	0.12%
Score Set X						0.69 %		0.42%

Table 1: Results of MS-ILS on MRP benchmark instances with $T_{max} = 300$ seconds

the average results and the best results. Also, very small scores are achieved on the largest instances with up to 50,000 processes. As such, the proposed method provides a valuable tool to optimize large scale multiprocessors systems.

To further analyze the performance of the method as the allowed CPU time grows, we conducted a set of 10 runs with different time limits, ranging from 30 seconds to 30 minutes. The average scores on each problem set are reported in Figure 3. From these results, it appears that the method is particularly well calibrated for achieving high-quality results in five minutes, with a score of 0.71% and 0.65% on the large-scale sets B and X, respectively. On the Set A again, the lack of good lower bound tends to complicate the analysis. Longer run times may lead to additional solution increases, but most of the possible improvement (more than 99% for sets B and X) is already achieved after five minutes.

Figure 3: Solution quality versus run time



CPU Time	30 s	60 s	2 min	5 min	10 min	30 min
Score Set A	8.06%	7.58%	7.35%	6.82%	6.61%	6.18%
Score Set B	11.29%	5.95%	3.07%	0.71%	0.50%	0.41%
Score Set X	11.76%	6.26%	3.00%	0.65%	0.44%	0.37%

5.5 MS-ILS performance on MCBPP benchmark instances

The performance of the method is also assessed on the classical MCBPP benchmark instances of Spieksma (1994) and Caprara and Toth (2001), using the problem transformation described in Section 4.4. Since these instances are smaller, the shaking phase is triggered whenever the LS has reached a local optimum, and 5000 shaking iterations without improvement of the current best solution are allowed before triggering a restart.

Table 2 presents the results of MS-ILS on 10 runs, and compare them with the current state-of-the-art solutions – Best Known Lower Bounds (BKLB) and Upper Bounds (BKUB) – reported by Monaci and Toth (2006) and, in the case of Class 10, the optimal number of bins known by instance construction. These results are aggregated by problem class, i.e. for each class the cumulated number of bins of the 10 instances is reported. Each column presents, in turn, the problem size, the problem class, the best known upper and lower bound, the average and the best number of bins generated with the MS-ILS metaheuristic, the number of BKUB found by MS-ILS, and the average CPU time per instance.

These experiments demonstrate the remarkable performance of the proposed MS-ILS approach on the classic MCBPP instances. All known best known upper bounds and optimal solutions obtained by previous algorithms have been found. For some of the largest instances of Class 10, some optimal solutions – known only from instance construction but not from any past algorithm – are not found. Nevertheless, finding these solutions is very difficult due to the way the instances were generated, without any unit of free resource in the optimal assignment (see discussion in Section 5.1).

13 new best upper bounds have been found. Ten of these solutions are for Class 1 with sizes 100 and 200, and three of these solutions are for Class 7 with size 100. These new upper bounds match the known lower bounds from Monaci and Toth (2006) and thus are now proven optimal.

Size	Class	BKLB	BKUB	ILS Avg	ILS Best	nb BKUB	Avg T(s)
25	1	69	69	69.0	69	10/10	12.7
25	6	101	101	101.0	101	10/10	21.3
25	7	96	96	96.0	96	10/10	18.6
25	9	73	73	73.0	73	10/10	20.3
24	10	80	80	80.0	80	10/10	11.1
50	1	135	135	135.0	135	10/10	72.3
50	6	214	215	215.0	215	10/10	68.6
50	7	196	197	197.0	197	10/10	88.0
50	9	144	145	145.0	145	10/10	199.2
51	10	170	170	170.0	170	10/10	68.9
100	1	255	260	257.0	257	10/10	294.5
100	6	405	410	410.0	410	10/10	300.0
100	7	398	405	402.1	402	10/10	285.9
100	9	257	267	267.0	267	10/10	300.0
99	10	330	330	330.0	330	10/10	232.4
200	1	503	510	503.0	503	10/10	300.0
200	6	803	811	812.4	811	10/10	300.0
200	7	799	802	802.0	802	10/10	300.0
200	9	503	513	513.0	513	10/10	300.0
201	10	670	670	679.8	678	2/10	300.0

Table 2: Performance of MS-ILS on the MCBPP instances

The best solutions of MS-ILS are reported in Table 3 in details in the column (ILS) along with the previous lower (LB) and upper bounds (UB). New optimal solutions are underlined.

6 Conclusions

This paper introduced an Iterated Local Search algorithm for Machine Reassignment and Multi-Capacity Bin Packing Problems. The proposed solution approach is very simple and relies on few parameters. It produces high-quality solutions in a reasonable computational time, even for large-scale instances of both problems. For the MRP, we can prove that in most cases more than 99% of the possible improvement from the initial solution have been achieved by reorganizing the processes with our method. For the MCBPP, all the current best known upper bounds and optimal solutions available from previous algorithms on classic benchmark instances have been retrieved, and 13 new best upper bounds have been produced. These new solutions match the lower bounds from the literature, and thus are optimal.

The MRP is a relatively new problem which opens many promising avenues of research. Future works involve developing new procedures for better managing the dependency relationships, better compound neighborhoods, exact components within the resolution, i.e. the so-called math-heuristics, and the use of lower bound information on sub-problems to guide the search. Further evaluation of the effectiveness of our optimization solution can be achieved by conducting cluster management simulations using traces of workloads provided by Google as input data (see Hellerstein (2010)). We may also consider implementing and evaluating our optimization solution in a real cluster management infrastructure, leveraging the work described by Petrucci et al. (2011).

		Class 1			**Class 6**			**Class 7**			**Class 9**			**Class 10**		
#	*Size*	LB	UB	ILS	LB	UB	ILS	LB	UB	ILS	LB	UB	ILS	LB	UB	ILS
1	S1	6	6	6	10	10	10	9	9	9	7	7	7	8	8	8
2		7	7	7	10	10	10	9	9	9	7	7	7	8	8	8
3		7	7	7	10	10	10	10	10	10	7	7	7	8	8	8
4		7	7	7	10	10	10	10	10	10	7	7	7	8	8	8
5		7	7	7	10	10	10	10	10	10	7	7	7	8	8	8
6		7	7	7	10	10	10	10	10	10	7	7	7	8	8	8
7		7	7	7	10	10	10	10	10	10	7	7	7	8	8	8
8		7	7	7	10	10	10	9	9	9	8	8	8	8	8	8
9		7	7	7	10	10	10	10	10	10	8	8	8	8	8	8
10		7	7	7	11	11	11	9	9	9	8	8	8	8	8	8
1	S2	13	13	13	21	21	21	21	21	21	14	14	14	17	17	17
2		13	13	13	21	21	21	18	18	18	14	14	14	17	17	17
3		13	13	13	21	21	21	21	21	21	14	14	14	17	17	17
4		13	13	13	21	21	21	18	18	18	14	14	14	17	17	17
5		13	13	13	21	21	21	21	21	21	14	14	14	17	17	17
6		14	14	14	22	22	22	18	18	18	14	15	15	17	17	17
7		14	14	14	21	22	22	21	22	22	15	15	15	17	17	17
8		14	14	14	22	22	22	18	18	18	15	15	15	17	17	17
9		14	14	14	22	22	22	22	22	22	15	15	15	17	17	17
10		14	14	14	22	22	22	18	18	18	15	15	15	17	17	17
1	S3	25	26	**25**	40	41	41	40	41	41	25	26	26	33	33	33
2		26	26	26	41	41	41	39	40	**39**	26	27	27	33	33	33
3		26	26	26	41	41	41	41	41	41	26	27	27	33	33	33
4		25	26	**25**	40	41	41	39	40	**39**	25	26	26	33	33	33
5		25	26	**25**	40	41	41	40	41	41	25	26	26	33	33	33
6		25	26	26	40	41	41	39	40	**39**	26	27	27	33	33	33
7		25	26	26	40	41	41	40	41	41	26	27	27	33	33	33
8		26	26	26	41	41	41	39	40	40	26	27	27	33	33	33
9		26	26	26	41	41	41	41	41	41	26	27	27	33	33	33
10		26	26	26	41	41	41	40	40	40	26	27	27	33	33	33
1	S4	50	51	**50**	80	81	81	80	80	80	50	51	51	67	67	68
2		50	51	**50**	80	81	81	80	80	80	50	51	51	67	67	68
3		50	51	**50**	80	81	81	80	80	80	50	51	51	67	67	68
4		50	51	**50**	80	81	81	80	80	80	50	51	51	67	67	68
5		50	51	**50**	80	81	81	80	80	80	50	51	51	67	67	68
6		50	51	**50**	80	81	81	79	80	80	50	51	51	67	67	68
7		50	51	**50**	80	81	81	80	81	81	50	51	51	67	67	67
8		51	51	51	81	81	81	79	80	80	51	52	52	67	67	68
9		51	51	51	81	81	81	81	81	81	51	52	52	67	67	68
10		51	51	51	81	82	82	80	80	80	51	52	52	67	67	67

For the instances of Class 1 to 9, (S1,S2,S3,S4) = (25,50,100,200).
For the instances of Class 10, (S1,S2,S3,S4) = (24,51,99,201).

Table 3: Detailed best solutions for the MCBPP instances

Acknowledgments

Partial funding for this project has been provided by the Champagne-Ardenne Regional Council, France, the French European Fund for Economic and Regional Development (FEDER), the Natural Sciences and Engineering Council of Canada (NSERC), the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT), the Brazilian Government agencies CNPq and CAPES. This support is gratefully acknowledged.

References

M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A View of Cloud Computing, Communications of the ACM 53 (4) (2010) 50–58.

Roadef, ROADEF/EURO Challenge 2012: Machine Reassignment, http://challenge.roadef.org/2012/en/index.php, last accessed 08/11/2012, 2012.

V. Petrucci, E. V. Carrera, O. Loques, J. Leite, D. Mossé, Optimized Management of Power and Performance for Virtualized Heterogeneous Server Clusters, in: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid (CCGrid'11), 2011.

A. Caprara, P. Toth, Lower bounds and algorithms for the 2-dimensional vector packing problem, Discrete Applied Mathematics 111 (3) (2001) 231–262.

R. Graham, Bounds on multiprocessing anomalies and related packing algorithms, in: AFIPS '72 (Spring): Proceedings of the Spring Joint Computer Conference, 1972.

L. T. Kou, G. Markowsky, Multidimensional bin packing algorithms, IBM Journal of Research and Development 21 (5) (1977) 443–448.

F. Spieksma, A branch-and-bound algorithm for the two-dimensional vector packing problem, Computers & Operations Research 21 (1) (1994) 19–25.

H. Kellerer, V. Kotov, An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing, Operations Research Letters 31 (1) (2003) 35–41.

S. Chang, H.-C. Hwang, S. Park, A two-dimensional vector packing model for the efficient use of coil cassettes, Computers & Operations Research 32 (8) (2005) 2051 – 2058.

M. Monaci, P. Toth, A Set-Covering-Based Heuristic Approach for Bin-Packing Problems, INFORMS Journal on Computing 18 (1) (2006) 71–85.

D. Wilcox, A. W. McNabb, K. D. Seppi, Solving virtual machine packing with a Reordering Grouping Genetic Algorithm., in: IEEE Congress on Evolutionary Computation, 362–369, 2011.

R. Panigrahy, K. Talwar, L. Uyeda, Heuristics for Vector Bin Packing, Tech. Rep., Microsoft, 2011.

H. Shachnai, T. Tamir, Approximation schemes for generalized two-dimensional vector packing with application to data placement, Journal of Discrete Algorithms 10 (2012) 35–48.

J. Beck, Modeling multicomputer task allocation as a vector packing problem, in: Proceedings of 9th International Symposium on Systems Synthesis, 115–120, 1996.

W. Leinberger, G. Karypis, Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints, in: ICPP '99 Proceedings of the 1999 International Conference on Parallel Processing, 404–412, 1999.

L. Epstein, T. Tassa, Vector assignment problems: a general framework, Journal of Algorithms 48 (2) (2003) 360–384.

J. L. Hellerstein, Google cluster data, Google research blog, posted at `http://googleresearch.blogspot.com/2010/01/google-cluster-data.html`., 2010.