Ann Oper Res (2016) 242:93–114
DOI 10.1007/s10479-014-1771-7

Simulated annealing for the machine reassignment problem

Gabriel M. Portal · Marcus Ritt · Leonardo M. Borba · Luciana S. Buriol

Published online: 13 January 2015

Abstract Given an initial assignment of processes to machines, the machine reassignment problem is to find an assignment that improves the machine usage, subject to several resource and allocation constraints, and considering reassignment costs. We propose a heuristic based on simulated annealing for solving this problem. It uses two neighborhoods, one that moves a process from one machine to another, and a second one that swaps two processes on different machines. We present data structures that permit to validate and execute a move in time $O(r + d)$ where r is the number of resources and d the number of dependencies of the service the process belongs to. The heuristic runs with two different sets of parameters in parallel until a convergence criterion is satisfied. The machine reassignment problem was subject of the ROADEF/EURO challenge in 2012, and the proposed algorithm ranked fourth in the final round of the senior category of the competition.

Keywords Machine reassignment · Scheduling · Simulated annealing

1 Introduction

Service providers like Google and Amazon need to manage a large computational infrastructure in order to provide the best quality of service to their clients. Naturally, they have a great interest in optimizing the usage of their resources, such as computing power, main memory,

G. M. Portal
Google Inc., Belo Horizonte, Minas Gerais, Brazil
e-mail: gabrielmportal@gmail.com

M. Ritt · L. M. Borba · L. S. Buriol (✉)
Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
e-mail: buriol@inf.ufrgs.br

M. Ritt
e-mail: marcus.ritt@inf.ufrgs.br

L. M. Borba
e-mail: lmborba@gmail.com

disk space, etc. However, there are other factors like the robustness, redundancy, and safety margins of resources of the system that must be considered. The impact of a machine failure must be minimized, and even if a complete data center shuts down it is desirable to keep the service running. Also, an overload of the system increases the probability of failure and should be avoided.

The steady growth of Internet, cloud computing and grid services also increased the importance of solving such large scale resource management problems. In this paper we solve a combinatorial optimization problem called the Machine Reassignment Problem (MRP), which models a specific resource management problem. Given a feasible assignment of processes to machines, the problem consists in finding an assignment that improves the overall machine usage. The usage of a machine is measured by costs representing the overall load, the resource balance, and the resource usage during the reassignment. Moreover, there are several hard constraints that should be considered by the assignment. A detailed description of the problem is given in the next section.

The MRP is an NP-hard combinatorial optimization problem, since it includes several NP-complete subproblems, e.g. bin packing, as special cases. A common decision for solving medium and large instances of such a problem is to use heuristic algorithms. We formulate the problem as an integer linear program and solve it with a general purpose solver, and propose and investigate a simulated annealing heuristic.

This problem was subject of the ROADEF/EURO challenge 2012 (Google Inc. 2011). The authors' team ranked fourth in the first and in the final phases of the challenge, among 27 finalists. A Variable Neighborhood Search (VNS) proposed by Gavranović et al. (2012) won the competition. It uses four neighborhoods: the *shift* neighborhood moves one process from one machine to another, the *swap* neighborhood exchanges two processes from different machines, the *chain* neighborhood shifts a subset of processes on given machines, and finally, the *BPR* (big process relocation) neighborhood shifts a single process to a new machine m and then other processes from m to a third machine. Mehta et al. (2012a) proposed two large neighborhood search procedures. They repeatedly solve a subproblem on selected machines and processes heuristically using integer linear programming or constraint programming. The approach using constraint programming performs better, and has subsequently been improved by tuning its parameters (Malitsky et al. 2013). Similarly, Brandt et al. (2012) proposed a large neighborhood search using constraint programming. Sansottera et al. (2012) run simulated annealing and VNS in parallel. Both heuristics use the shift and swap neighborhoods. The VNS shakes a locally optimal solution for both neighborhoods in a randomized k-step neighborhood, followed by a sample local search. Teypaz (2012) as well as Pécot (2012) proposed a tabu search using the shift and swap neighborhoods. Teypaz (2012) restricts the moves to the best shifts and swaps between selected under- and overused machines, determined by a maximum weight matching. Pécot (2012) alternates between the two neighborhoods every few steps, and limits the exploration of the neighborhood by sampling. Both approaches use the tabu mechanism to prohibit move reversal for a fixed number of iterations. Masson et al. (2012) proposed a multi-start iterated local search (ILS) that uses the shift and the swap neighborhoods. The swap neighborhood can also exchange one process by two other processes. Since exploring all neighbors can demand considerable time, considering both neighborhoods, only a randomly-selected subset of neighbors is explored. The size of the subset depends on an input parameter. Finally, Lopes et al. (2014) propose some variants of an ILS algorithm based on the shift and swap neighborhoods. Again reduced neighborhoods are explored. However, instead of evaluating random neighbors, processes and machines are sorted in such way that promising neighbors are explored first. The variants rely on IP-based and randomized perturbations. The IP-based perturbation fixes a subset of variables, and then uses CPLEX to solve subproblems restricted to a run of 5 s.

The remainder of this paper is organized as follows. Section 2 introduces the problem and Sect. 3 gives a mathematical formulation. Section 4 presents the proposed simulated annealing heuristic. A lower bound is proposed in Sect. 5. Computational results are reported in Sect. 6, and we conclude in Sect. 7.

2 Problem definition and notation

In the MRP, the main elements are a set of machines $\mathcal{M}$ and a set of processes $\mathcal{P}$. A solution of the problem is an assignment $A : \mathcal{P} \mapsto \mathcal{M}$ of the processes to machines. For example, $A(p) = m$ means that, in solution A, process p has been assigned to machine m. A feasible initial assignment A_0 is part of a problem instance. The objective is to improve the machine usage by finding a better assignment and attending several constraints that are presented in this section.

Each machine has available resources. Resources are abstract quantities, and may represent, for example, the processing capacity or the total memory of a machine. Secondary storage capacity is also an important resource that could be considered. Each machine disposes of a certain amount of each resource, and each process consumes part of each resource. Thus, there are problem constraints that assure the resource capacities of the machines are respected. Formally, $\mathcal{R}$ is the set of resources, $\bar{U}_{mr}$ is the capacity of resource $r \in \mathcal{R}$ in machine $m \in \mathcal{M}$, and R_{pr} is the requirement of process $p \in \mathcal{P}$ for resource $r \in \mathcal{R}$.

When assigning a process to a new machine, some additional resources may be required during the relocation. A resource with *transient usage* is consumed on the source and destination machines while a process is transferred. An example of this situation occurs with secondary storage: during the relocation of a process between machines, disk space may be used (or reserved) on both machines. Formally, $\mathcal{TR} \subseteq \mathcal{R}$ is the subset of the resources that need transient usage.

The processes provide different *services*, and each process is assigned to a single service. Processes of the same service have a common objective, for example providing an email or a localization service. There are several processes, spread over machines, that cooperate to deliver a high quality service to clients. Services are represented by a partition $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\}$ of the set of processes $\mathcal{P}$ (i.e. $\cup_{s \in \mathcal{S}} s = \mathcal{P}$, $s_i \cap s_j = \emptyset$ for $i \neq j$). Similarly, each machine has a specific *location*. Locations represent the geographical distribution of the machines. The locations are given by a partition $\mathcal{L}$ of the set of machines $\mathcal{M}$, each location being composed of one or more machines. Finally, each machine is part of a *neighborhood*. Neighborhoods define groups of machines that can easily communicate. Two machines are part of the same neighborhood if they can exchange data fast enough to answer the requests of an user in an acceptable time. In practice neighborhoods will often be a refinement of locations, since typical servicing times are so small that two machines must be in the same geographical location to be able to cooperate in the provision of services. Neighborhoods are given by a partition $\mathcal{N}$ of the set of machines $\mathcal{M}$.

2.1 Hard constraints

Hard constraints must be respected by any feasible solution of the problem. In contrast, soft constraints can be violated, but are penalized in the objective function. Next, the five hard constraints of the MRP are presented.

2.1.1 Capacity constraints

The capacity constraints guarantee that the usage of the resources of a machine does not exceed its capacity. For an assignment A, let $A^{-1}(m)$ be the set of processes assigned under A to machine $m \in \mathcal{M}$ and

$$U_{mr} = \sum_{p \in A^{-1}(m)} R_{pr} \tag{1}$$

be the usage of resource $r \in \mathcal{R}$ by machine $m \in \mathcal{M}$. Then the capacity constraints are

$$U_{mr} \leq \bar{U}_{mr}, \qquad \forall m \in \mathcal{M}, \quad \forall r \in \mathcal{R}. \tag{2}$$

2.1.2 Transient usage constraints

Transient usage constraints complement the capacity constraints, taking into account the additional usage of some resources during the relocation of processes between machines. For example, it is not possible to free the disk space consumed by some process on the source machine before the content is copied to the target machine. This space, however, is already reserved on the target.

The capacity constraints of transient usage resources strengthen the capacity constraints (2): the resources consumed on a machine $m \in \mathcal{M}$ are the sum of the resource requirements of processes assigned to m under the initial assignment A_0, and the resource requirements of processes assigned to m under the new assignment A. We thus have

$$\sum_{p \in A^{-1}(m) \cup A_0^{-1}(m)} R_{pr} \leq \bar{U}_{mr}, \qquad \forall m \in \mathcal{M}, \quad r \in \mathcal{TR}.$$

2.1.3 Conflict constraints

Conflict constraints prohibit to assign two processes of the same service to the same machine. They minimize the impact on a single service if some machine fails. The conflict constraints can be expressed as

$$A(p) \neq A(p'), \qquad \forall s \in \mathcal{S}, \quad p \in s, \quad p' \in s.$$

2.1.4 Spread constraints

Spread constraints guarantee that the processes of a service are sufficiently spread among the machine locations. Thus, in case all machines of a location fail, e.g. due to a natural catastrophe or a focused attack, the services will still be able to process requests. Let $\mathrm{sp}_s^{\min}$ be the minimal number of locations that must be occupied by processes of service $s \in \mathcal{S}$, and $\mathrm{sp}_{s,l}$ be the set of processes of service $s \in \mathcal{S}$ in location $l \in \mathcal{L}$. We have

$$\mathrm{sp}_{s,l} = \{p \in s \mid A(p) \in l\}, \qquad \forall s \in \mathcal{S}, \quad l \in \mathcal{L},$$

and a feasible assignment must satisfy the spread constraints

$$\sum_{l \in \mathcal{L}} \min(1, |sp_{s,l}|) \geq \mathrm{sp}_s^{\min}, \qquad \forall s \in \mathcal{S}.$$

2.1.5 Dependency constraints

Dependency constraints guarantee the efficiency of communication among processes of one service and processes of a dependent service. If service s depends on service s', then for each process of s in some neighborhood, there must be at least one process of s' in the same neighborhood. This ensures that there always exists a process of service s' that can efficiently answer requests of service s. Formally, let $s \to s'$ indicate that service s depends on service s', and denote the neighborhood of process p by $\mathrm{N}(p)$. Then the dependency constraints are

$$\forall s, s' \in \mathcal{S} \mid s \to s',\ \forall p \in s,\ \exists p' \in s' \mid \mathrm{N}(p) = \mathrm{N}(p').$$

2.2 Objective function

The objective function penalizes several soft constraints, that are not mandatory but define desirable characteristics of a solution. It is the sum of the load cost C^L, the balance cost C^B, the process move cost C^P, the machine move cost C^M, and the service move cost C^S.

2.2.1 Load cost

Although a machine has a maximum capacity for a resource, it is desirable to keep the usage of the resource lower for a better operation. For example, if the memory of a machine is completely used, any new operation assigned to it would exceed the available capacity and degrade its performance.

Let $U^s_{mr} \le \bar{U}_{mr}$ be the safety capacity of machine $m \in \mathcal{M}$ for resource $r \in \mathcal{R}$. It is desirable that the usage of the resource r in machine m is less than U^s_{mr}. Any excess over the safety capacity is penalized linearly.

The load cost $l \in \mathbb{R}^{|\mathcal{R}|}$ is defined as

$$l_r = \sum_{m \in \mathcal{M}} \max(0, U_{mr} - U^s_{mr}). \tag{3}$$

For each resource $r \in \mathrm{R}$, and for weights $w^L \in \mathbb{R}^{|\mathcal{R}|}$ the total load cost is

$$C^L = w^L l. \tag{4}$$

2.2.2 Balance cost

The balance cost term of the objective function measures the balance of resources in the machines, which is especially important for future usage of machines. It is not useful, for example, to have free memory space on a machine if there are no processing resources available. The balance cost defines target ratios of the usage for pairs of resources and penalizes deviations.

Let $\mathcal{D} \subseteq \mathcal{R}^2 \times \mathbb{N}$ be the set of dependencies between resources. A dependency $d = (r_1(d), r_2(d), t(d)) \in \mathcal{D}$ is a triplet of two resources and a target ratio, and penalizes consumption of resource $r_1(d)$ that exceeds $1/t(d)$ times the consumption of resource $r_2(d)$. The set of dependencies is part of the problem instance. Let $F_{mr} = \bar{U}_{mr} - U_{mr}$ be the free capacity of resource $r \in \mathcal{R}$ on machine $m \in \mathcal{M}$. Then the balance cost for a given dependency $d \in \mathcal{D}$ is

$$b_d = \sum_{m \in \mathcal{M}} \max(0, t(d) F_{m,r_1(d)} - F_{m,r_2(d)}), \tag{5}$$

and for a weight vector $w^B \in \mathbb{R}^{|\mathcal{D}|}$ the total balance cost is

$$C^B = w^B b. \tag{6}$$

2.2.3 Process move cost

Moving processes to new machines is undesirable because the processes will not be able to answer requests during the transfer. Besides this, some processes may be more costly to move than others, for example, due to the amount of data to be transferred. Let c^P_p be the cost to move process $p \in \mathcal{P}$, and w^P be the weight of the process move cost in the objective function. Then, the total process move cost is

$$C^P = w^P \sum_{p \in \mathcal{P} | A(p) \neq A_0(p)} c^P_p. \tag{7}$$

2.2.4 Machine move cost

Moving a process between machines that are in the same room may be different from moving a process between machines in different continents. At least, the network speed is different, thus the transfer time is affected. Let $c^M_{mm'}$ be the cost of transferring a process from machine m to machine m'. Naturally, $c^M_{m,m} = 0$ for every machine $m \in \mathcal{M}$. Also, let w^M be the weight of the machine move cost in the objective function. The total machine move cost is

$$C^M = w^M \sum_{p \in \mathcal{P}} c^M_{A_0(p), A(p)}. \tag{8}$$

2.2.5 Service move cost

The service move cost measures how the transfer of processes affects individual services. This cost reflects the maximum number of moved processes of a service, over all services. Let c^S_s be the number of processes of service $s \in \mathcal{S}$ that have been assigned to a new machine. We have

$$c^S_s = |\{p \in s \mid A(p) \neq A_0(p)\}|. \tag{9}$$

Let w^S be the weight of the service move cost. The service move cost is

$$C^S = w^S \max_{s \in \mathcal{S}} c^S_s. \tag{10}$$

3 Mathematical formulation of the MRP

In this section we present an integer linear program for the MRP. Its uses decision variables $x_{pm} \in \{0, 1\}$, where $x_{pm} = 1$ if and only if process $p \in \mathcal{P}$ is assigned to machine $m \in \mathrm{M}$, and several auxiliary variables. Table 1 lists all auxiliary variables used in the formulation. The MRP can be formulated as

Table 1 Auxiliary variables used in the IP formulation of the MRP

Quantity	Description
$y_p \in \{0, 1\}$	Indicates whether the process $p \in \mathcal{P}$ was reassigned
$U_{mr} \in \mathbb{R}_+$	Usage of machine $m \in \mathcal{M}$ for resource $r \in \mathcal{R}$
$A_{sl} \in \{0, 1\}$	Indicates whether service $s \in \mathcal{S}$ has at least one process in location $l \in \mathcal{L}$
$B_{sn} \in \{0, 1\}$	Indicates whether service $s \in \mathcal{S}$ has at least one process in neighborhood $n \in \mathcal{N}$
$l_{mr} \in \mathbb{R}_+$	Load cost of machine $m \in \mathcal{M}$ for resource $r \in \mathcal{R}$
$b_{dm} \in \mathbb{R}_+$	Balance cost for dependency $d \in \mathcal{D}$ and machine $m \in \mathcal{M}$
$C^L \in \mathbb{R}_+$	Total load cost
$C^B \in \mathbb{R}_+$	Total balance cost
$C^P \in \mathbb{R}_+$	Total process move cost
$C^S \in \mathbb{R}_+$	Total service move cost
$C^M \in \mathbb{R}_+$	Total machine move cost

$$\min. \quad C^L + C^B + C^P + C^M + C^S, \tag{11}$$

$$\text{s.t.} \quad \sum_{m \in \mathcal{M}} x_{pm} = 1, \qquad \forall p \in \mathcal{P}, \tag{12}$$

$$\sum_{p \in s} x_{pm} \le 1, \qquad \forall s \in \mathcal{S}, m \in \mathcal{M}, \tag{13}$$

$$y_p = 1 - x_{p, A_0(p)}, \qquad \forall p \in \mathcal{P}, \tag{14}$$

$$U_{mr} = \sum_{p \in \mathcal{P}} x_{pm} R_{pr}, \qquad \forall m \in \mathcal{M}, r \in \mathcal{R}, \tag{15}$$

$$U_{mr} \le \bar{U}_{mr}, \qquad \forall m \in \mathcal{M},\ \forall r \in \mathcal{R} \setminus \mathcal{TR}, \tag{16}$$

$$U_{mr} + \sum_{p \in A_0^{-1}(m)} y_p R_{pr} \le \bar{U}_{mr}, \qquad \forall m \in \mathcal{M}, r \in \mathcal{TR}, \tag{17}$$

$$A_{sl} \le \sum_{m \in l, p \in s} x_{pm}, \qquad \forall s \in \mathcal{S}, l \in \mathcal{L}, \tag{18}$$

$$\sum_{l \in \mathcal{L}} A_{sl} \ge \text{sp}_s^{\min}, \qquad \forall s \in \mathcal{S}, \tag{19}$$

$$\sum_{m \in n, p \in s} x_{pm} / \min\{|n|, |s|\} \le B_{sn}, \qquad \forall s \in \mathcal{S}, n \in \mathcal{N}, \tag{20}$$

$$B_{sn} \le \sum_{m \in n, p \in s} x_{pm}, \qquad \forall s \in \mathcal{S}, n \in \mathcal{N}, \tag{21}$$

$$B_{sn} \le B_{tn}, \qquad \forall n \in \mathcal{N}, s \to t, \tag{22}$$

$$l_{mr} \ge U_{mr} - U^s_{mr}, \qquad \forall m \in \mathcal{M}, r \in \mathcal{R}, \tag{23}$$

$$C^L = \sum_{m \in \mathcal{M}, r \in \mathcal{R}} w_r^L l_{mr}, \tag{24}$$

$$\begin{aligned} b_{dm} \ge\ & t(d)(\bar{U}_{m,r_1(d)} - U_{m,r_1(d)}) \\ & - (\bar{U}_{m,r_2(d)} - U_{m,r_2(d)}), \qquad \forall d \in \mathcal{D}, m \in \mathcal{M}, \end{aligned} \tag{25}$$

$$C^B = \sum_{m \in \mathcal{M}, d \in \mathcal{D}} w_d^B b_{mb}, \tag{26}$$

$$c_s^S \geq \sum_{p \in s} y_p, \qquad \forall s \in \mathcal{S}, \tag{27}$$

$$C^S \geq w^S \sum_{s \in \mathcal{S}} c_s, \tag{28}$$

$$C^P = w^P \sum_{p \in \mathcal{P}} y_p c_p^P, \tag{29}$$

$$C^M = w^M \sum_{p \in \mathcal{P}, m \in \mathcal{M}} x_{pm}\, c^M_{A_0(p),m}. \tag{30}$$

Constraints (12) and (13) ensure that each process is assigned to exactly one machine, and there is no conflict, i.e. no two processes of the same service are assigned to the same machine. Auxiliary variables y are defined by (14). Constraints (15)–(17) assert that the static and transient resource requirements do not exceed their capacity. Constraints (18) force A_{sl} to be 0, if there is no process from service s in location l. Otherwise, it is free to assume any value, and is set to 1 as required to satisfy the spread constraints (19). Similarly, B_{sn} indicates the presence of service s in neighborhood n and is defined by (20) and (21), and used in the dependency constraints (22). Constraints (20) force sets B_{sn} to 1 if there is at least one service from s into neighborhood n, while constraints (21) set B_{sn} to 0 otherwise. The remaining constraints are the soft constraints which enter into the objective function. The auxiliary variable l_{mr} in (23) is the load cost of resource r on machine m, and Eq. (24) defines total load cost. Similarly, the balance cost for dependency d and machine m is defined in constraints (25), Eq. (26) gives the total balance cost, constraints (27) define the number of moved processes in service s, and Eq. (28) defines the overall service move cost. Finally, Eqs. (29) and (30) define the process and machine move cost.

4 Simulated annealing applied to the MRP

In this section we propose a heuristic based on simulated annealing for the MRP. Simulated annealing was proposed by Kirkpatrick et al. (1983), and independently by Cerny (1985). It is a stochastic search procedure, which repeatedly selects a random neighbor of the current solution, and accepts it as the new solution if it improves the current one, or with probability $\exp(-\Delta/T)$ if it is Δ worse than the current solution. The parameter T is called the *temperature*. For very high temperatures, the search effectively is a random walk in the solution space, while for very low temperatures it is a stochastic local search. The temperature follows a *cooling cycle* that starts with a sufficiently high value and decreases slowly. It is well known that for a sufficiently slow cooling schedule simulated annealing converges to the optimal solution (Hajek 1988). A heuristic based on simulated annealing is defined by a suitable choice of a neighborhood and a careful calibration of the parameters: convergence tends to be slow and time spent in the random walk or local search domains usually is ineffective (Johnson et al. 1989).

Our approach uses the shift and swap neighborhoods explained above. We combine both neighborhoods: with probability α, we select a feasible random neighbor from the shift neighborhood, and with probability $1 - \alpha$ one from the swap neighborhood.

For an instance with $p = |\mathcal{P}|$ processes and $m = |\mathcal{M}|$ machines the size of these neighborhoods is $O(pm)$ and $O(p^2)$, respectively. For the size of the instances we are interested in ($p \approx 20{,}000$ and $m \approx 1{,}000$) it is inefficient to uniformly choose a feasible neighbor in the current neighborhood. We therefore propose a more efficient sampling strategy. For the move neighborhood, a process p and a machine k are selected uniformly at random. Then, we consider machines $k + i \mod m$ for $0 \leq i \leq c$, where c is a constant. The first valid assignment of process p to a machine in the given order, if any, is chosen. Otherwise, process p is considered unmovable and the move is rejected. A similar procedure is used for the swap neighborhood: a process is fixed and a sequence of c other processes is chosen to perform the movement. This strategy guarantees an efficient choice of the neighbor in constant time.

We choose a *geometric cooling cycle* that starts with an initial temperature t_0, holds the temperature constant for n iterations, and then reduces it with a cooling rate r. When the current best solution value is not updated for $20n$ iterations and the number of accepted moves is less than 0.1 % we consider the solution of the current cooling cycle "frozen" (Johnson et al. 1989). In this case, we reheat by increasing the temperature to $t_0/100$. The objective of this reheating procedure is to perform more significant perturbations to the current solution, aiming to escape from a local minimum.

4.1 Speeding up computation by using suitable data structures

The MRP has several soft and hard constraints. Verifying the feasibility of a solution and evaluating the soft constraints when computing the objective function demand a high computational effort. Thus, choosing a suitable set of data structures is important to reduce the overall computational time.

There are some core operations that occur in any local search algorithm for this problem. For example, we must verify if a movement of the neighborhood is valid, compute the cost of the neighbor solution, and finally execute the move. In order to speedup these operations, we introduce a set of additional, redundant data structures:

(a) *machineResource* a matrix of $|\mathcal{M}||\mathcal{R}|$ integers that stores the usage of each resource in each machine. The current usage of resource $r \in \mathcal{R}$ on machine $m \in \mathcal{M}$ is $\text{machineResource}_{mr}$.
(b) *transientUsage* a matrix of $|\mathcal{M}||\mathcal{R}|$ integers that stores the usage of transient resources in machines. The usage of resource $r \in \mathcal{R}$ in machine $m \in \mathcal{M}$ by processes that were originally assigned to machine m and later moved to another machine is $\text{transientUsage}_{mr}$.
(c) *serviceMachine* this matrix of $|\mathcal{S}||\mathcal{M}|$ Boolean values is used for indicating if a service has any process assigned to a given machine. Therefore, $\text{serviceMachine}_{sm}$ indicates the existence of a process from service $s \in \mathcal{S}$ in machine $m \in \mathcal{M}$. Note that at most one process of a service may be assigned to a machine at a given time, so this matrix is boolean.
(d) *serviceLocation* a matrix of $|\mathcal{S}||\mathcal{L}|$ integers; $\text{serviceLocation}_{sl}$ equals the number of processes of service $s \in \mathcal{S}$ assigned to machines of location $l \in \mathcal{L}$.
(e) *serviceNeighborhood* a matrix of $|\mathcal{S}||\mathcal{N}|$ integers. The number of processes of service $s \in \mathcal{S}$ present in machine of neighborhood $n \in \mathcal{N}$ $\text{serviceNeighborhood}_{sn}$.
(f) *serviceLocationCount* a vector of $|\mathcal{S}|$ integers used to store the current spread factor of services. Value $\text{serviceLocationCount}_s$ equals the number of locations that have processes of service $s \in \mathcal{S}$.
(g) *serviceChanged* a vector of $|\mathcal{S}|$ integers. Value serviceChanged_s equals the number of processes of service $s \in \mathcal{S}$ that are not assigned to their original machines.

(h) *serviceChangedCount* $\text{serviceChangedCount}_k$ is the number of services that have exactly k processes which changed from their original machines. The size of this vector is $|\mathcal{P}|$ integers, but it could be the size of the biggest service.
(i) *maxServiceChanges* this is an integer value indicating the maximum number of processes of the same service that changed from machine. It is the service move cost of the solution before multiplying it by w^S.

The total memory usage of these data structures is $2|\mathcal{M}||\mathcal{R}| + |\mathcal{M}||\mathcal{S}| + |\mathcal{S}||\mathcal{L}| + |\mathcal{S}||\mathcal{N}| + 2|\mathcal{S}| + |\mathcal{P}|$ integers. It will usually be dominated by the term $|\mathcal{M}||\mathcal{S}|$. The ROADEF/EURO challenge limits the instance size to 50,000 processes, 5,000 machines and services, 1,000 locations and neighborhoods, 20 resources. Within these limits the total memory requirement never exceeds 35 M integers, or 135 MB on 32-bit machines.

Suppose we are moving process $p \in \mathcal{P}$ to machine $m \in \mathcal{M}$. Let s be the service of process p and l be the location of machine m. Then the conflict constraints are satisfied if $\text{serviceMachine}_{s,m} = 0$. Using matrices machineResource and transientUsage, we can verify in $O(|\mathcal{R}|)$ if the capacity constraints are satisfied. The spread constraints can be verified in constant time by making sure that the move will not lower the spread of service s below its minimum requirements. This is the case if $\text{serviceLocationCount}_s > \text{spread}_s$ or, if $\text{serviceLocationCount}_s = \text{spread}_s$, if p is not the only process of service s in its current location, or p is the first process of service s in location l. The most complex constraint is the dependency constraint. Let $n \in \mathcal{N}$ be the neighborhood of machine m. We must guarantee that n has processes that satisfy all the dependencies of service s. Besides this, process p cannot be moved if it is the only process of service p in its current neighborhood and another process depends on it. For the set of dependencies $\mathcal{D}$, all these requirements can be verified in $O(|\mathcal{D}|)$. In summary, the feasibility of a shift move can be verified in $O(|\mathcal{R}| + |\mathcal{D}|)$.

The difference between the cost of the new solution and the cost of the current solution can also be computed in $O(|\mathcal{R}| + |\mathcal{D}|)$. The load cost can be found in $O(|\mathcal{R}|)$ by verifying the state of the source and destination machines. The balance cost is computed in $O(|\mathcal{D}|)$ by verifying the state of the involved machines in case the move is performed. Move costs can be computed in constant time. When executing the move, all data structures must be updated to reflect the new assignment. This can be done in $O(|\mathcal{R}|)$. This cost is due to the update of the matrices machineResource and transientUsage. All other data structures can be updated in constant time. By a very similar procedure, the move in the swap neighborhood can be verified to be feasible in $O(|\mathcal{R}| + |\mathcal{D}|)$ and executed in $O(|\mathcal{R}|)$.

5 A lower bound for the MRP

In this section we propose a lower bound for the MRP. It serves to give an optimality gap for the solution. This is particularly important for the large instances, since the linear relaxation of the mathematical formulation could not be solved in reasonable time.

5.1 Load cost lower bound

The load cost lower bound LB^L is calculated as follows. For each resource, we consider the total capacity of the machines and the total usage of the processes. Basically, the constraint of assignment of a process to a single machine is relaxed: the resources of a process may be assigned to many machines and one resource of a process may also be divided among several machines. We have

$$LB^L = \sum_{r\in\mathcal{R}} w_r^L \max\left(0, \sum_{p\in\mathcal{P}} R_{pr} - \sum_{m\in\mathcal{M}} U_{mr}^s\right). \tag{31}$$

This is a very simple bound that relies on the following observation. If we take, for every resource, the sum of the requirements over all processes and the sum of the safety capacities over all machines, the excess of resource requirements is a lower bound for the load cost. However, when solving the original problem, the excess of resources over the safety capacities could be much higher.

Proposition 1 *The proposed load cost lower bound is at most the total load cost of any feasible solution, i.e,* $LB^L \leq C^L$.

Proof We have

$$\begin{aligned} C^L &= \sum_{r\in\mathcal{R}} w_r^L \sum_{m\in\mathcal{M}} \max(0, U_{mr} - U_{mr}^s) \\ &\geq \sum_{r\in\mathcal{R}} w_r^L \max\left(0, \sum_{m\in\mathcal{M}} U_{mr} - U_{mr}^s\right) \\ &= \sum_{r\in\mathcal{R}} w_r^L \max\left(0, \sum_{m\in\mathcal{M}} U_{mr} - \sum_{m\in\mathcal{M}} U_{mr}^s\right) \\ &= \sum_{r\in\mathcal{R}} w_r^L \max\left(0, \sum_{p\in\mathcal{P}} R_{pr} - \sum_{m\in\mathcal{M}} U_{mr}^s\right). \end{aligned}$$

□

5.2 Balance cost lower bound

A lower bound for the balance cost can be obtained in a similar way as the lower bound for the load cost. Instead of analyzing each machine separately for verifying the difference of free space for the considered resources, one considers the free space for all machines and the requirements over all processes. Let us call this lower bound LB^B, then we have

$$LB^B = \sum_{d\in\mathcal{D}} w_b^B \max(0, t(d)\, E(r_1(d)) - E(r_2(d))), \tag{32}$$

where $E(r)$ is the excess of the total capacity for resource r over the total requirements for this resource, defined as

$$E(r) = \sum_{m\in\mathcal{M}} \bar{U}_{mr} - \sum_{p\in\mathcal{P}} R_{pr}. \tag{33}$$

Moreover, $\bar{U}_{mr}$ is the capacity of machine m for resource r, $t(d)$, $r_1(d)$, and $r_2(d)$ are the target factor, the first resource, and the second resource of dependency d, respectively, as defined in Sect. 2.2.2, and w_b^B is the weight for this balance cost.

Proposition 2 *The proposed balance cost lower bound is at most the total balance cost of any feasible solution, i.e,* $LB^B \leq C^B$.

Proof First, observe that

$$\sum_{m\in\mathcal{M}} F_{m,r} = \sum_{m\in\mathcal{M}} \bar{U}_{mr} - U_{mr} = \sum_{m\in\mathcal{M}} \bar{U}_{mr} - \sum_{m\in\mathcal{M}} U_{mr} = E(r).$$

Thus we have

$$\begin{aligned} \mathrm{C}^{\mathrm{B}} &= \sum_{d\in\mathcal{D}} w_d^B \sum_{m\in\mathcal{M}} \max(0, t(d)\ F_{m,r_1(d)} - F_{m,r_2(d)}) \\ &\geq \sum_{d\in\mathcal{D}} w_d^B \max(0, t(d) \sum_{m\in\mathcal{M}} F_{m,r_1(d)} - \sum_{m\in\mathcal{M}} F_{m,r_2(d)}) \\ &= \sum_{d\in\mathcal{D}} w_d^B \max(0, t(d)\ E(r_1(d)) - E(r_2(d))) \end{aligned}$$

□

5.3 Total lower bound

The final lower bound is the sum of the load cost lower bound and balance cost lower bound, hence

$$LB = LB^L + LB^B. \tag{34}$$

Theorem 1 *The proposed lower bound is valid, i.e, $LB \leq C$.*

Proof Combining Eq. (34) with Propositions 1 and 2 we have

$$C \geq LB^L + LB^B + C^P + C^M + C^S \geq LB^L + LB^B = LB$$

since all costs are non-negative. □

The proposed lower bound might not be very strong because it does not take into account move costs and does not consider the combined load cost and balance cost. However, it presented good results for many of the instances, as shown in Sect. 6.3. A tighter lower bound could be achieved solving a relaxed problem considering both costs simultaneously.

6 Computational results

6.1 Test instances

The data set is composed of three instance groups A, B, and X of 10 instances each, made available during the ROADEF/EURO Challenge. Group A was released before and group B after the qualification phase. Groups B and X were the basis for the final comparison of the competitors. Table 2 shows the details of the instances: the number of processes $|\mathcal{P}|$, machines $|\mathcal{M}|$, resources $|\mathcal{R}|$ and transient resources $|\mathcal{TR}|$, services $|\mathcal{S}|$, locations $|\mathcal{L}|$, neighborhoods $|\mathcal{N}|$, dependencies, and the number of balance costs $|\mathcal{D}|$.

All instances have the same weights $w^P = 1$, $w^S = 10$, $w^M = 100$, $w_r^L = w_d^B = 10$ for all $r \in R$ and $d \in D$, the same process move costs $c_p^P = 1$, and machine move costs $c_{p,p'}^M \in \{0, 1, 2\}$.

Table 2 Characteristics of the instances

Inst.	$\|\mathcal{P}\|$	$\|\mathcal{M}\|$	$\|\mathcal{R}\|$	$\|\mathcal{TR}\|$	$\|\mathcal{S}\|$	$\|\mathcal{L}\|$	$\|\mathcal{N}\|$	Dep.	$\|\mathcal{D}\|$
A1-1	100	4	2	0	79	4	1	0	1
A1-2	1,000	100	4	1	980	4	2	40	0
A1-3	1,000	100	3	1	216	25	5	342	0
A1-4	1,000	50	3	1	142	50	50	297	1
A1-5	1,000	12	4	1	981	4	2	32	1
A2-1	1,000	100	3	0	1,000	1	1	0	0
A2-2	1,000	100	12	4	170	25	5	0	0
A2-3	1,000	100	12	4	129	25	5	577	0
A2-4	1,000	50	12	0	180	25	5	397	1
A2-5	1,000	50	12	0	153	25	5	506	1
B-1	5,000	100	12	4	2,512	10	5	4,412	0
B-2	5,000	100	12	0	2,462	10	5	3,617	1
B-3	20,000	100	6	2	15,025	10	5	16,560	0
B-4	20,000	500	6	0	1,732	50	5	40,485	1
B-5	40,000	100	6	2	35,082	10	5	14,515	0
B-6	40,000	200	6	0	14,680	50	5	42,081	1
B-7	40,000	4,000	6	0	15,050	50	5	43,873	1
B-8	50,000	100	3	1	45,030	10	5	15,145	0
B-9	50,000	1,000	3	0	4,609	100	5	43,437	1
B-10	50,000	5,000	3	0	4,896	100	5	47,260	1
X-1	5,000	100	12	4	2,529	10	5	4,164	0
X-2	5,000	100	12	0	2,484	10	5	3,742	1
X-3	20,000	100	6	2	14,928	10	5	15,201	0
X-4	20,000	500	6	0	1,190	50	5	38,121	1
X-5	40,000	100	6	2	34,872	10	5	20,560	0
X-6	40,000	200	6	0	14,504	50	5	39,890	1
X-7	40,000	4,000	6	0	15,273	50	5	43,726	1
X-8	50,000	100	3	1	44,950	10	5	12,150	0
X-9	50,000	1,000	3	0	4,871	100	5	45,457	1
X-10	50,000	5,000	3	0	4,615	100	5	47,768	1

6.2 Experimental methodology

All results have been obtained on a PC with an AMD FX-8150 3.6 GHz processor and 12 GB of main memory, running a 64-bit Linux operation system (Ubuntu 12.04). The method was implemented in C++ and compiled with the `gcc` compiler, version 4.7.3 with optimization flag −03. The random number generator is the Boost implementation of the Mersenne twister (Matsumoto and Nishimura 1998). The mathematical models were solved using CPLEX 12.5 running with two threads. We compare our results to those of the six best participants of the competition, which made their source code available. Their implementations have been compiled and run in the same environment, and with their native degree of parallelism. For comparison of scores under the conditions of the challenge we ran each

Table 3 Implementations used in the computational experiments

Team	Method	Parallelism
S41	VNS (Gavranović et al. 2012)	2
S38	Constraint programming (Mehta et al. 2012b)	1
S23	Simulated annealing (this paper)	2
J25	Constraint programming (Brandt et al. 2012)	2
J33	VNS & Simulated annealing (Sansottera et al. 2012)	2
S14	Tabu search (Teypaz 2012)	2
S34	Tabu search (Pécot 2012)	4

method once with a seed fixed to 9,999. We also compare the relative deviations from the best known values of all methods. In these experiments we obtained 20 replications for our method, to analyze its robustness. An overview of the methods is given in Table 3.

We systematically tested combinations of the parameters applied to the subset of instances A1-4, A2-1, A2-2, A2-3, A2-5, B-1, and B-3, since we considered these instances to be the most difficult for our method. The values of parameters we tested were $n \in \{10^4, 10^5, 10^6\}$, $r \in \{0.91, 0.95, 0.97\}$, $\alpha \in \{0, 0.3, 0.5, 0.7, 1.0\}$, $c \in \{25, 50, 100, 200, 400\}$ and $t_0 \in \{10^7, 10^8, 10^9\}$. All combinations of these values were tested, and for each parameter setting and instance, we ran four executions with different seeds. We next computed the average relative deviation from the best known values obtained by each pair of parameter settings. For each such pair, the best of the two values obtained for a fixed seed was used to compute the relative deviation. This calibration is targeted at the conditions of the challenge, where two parallel threads could be used, and allows to choose the best complementary pair of parameter settings.

The pair of parameter settings which performed best was

$$n = 10^5, \quad r = 0.97, \quad \alpha = 0.7, \quad c = 50, \quad t_0 = 10^7,$$

and

$$n = 10^6, \quad r = 0.95, \quad \alpha = 0.7, \quad c = 25, \quad t_0 = 10^8.$$

The first parameter setting was also the one that performed best for a single thread. We have further tested the robustness of these parameter settings, by evaluating the change in the average relative deviation from the best known values, when a single parameter is changed to its next higher or lower level, if possible. We found that the parameter setting is robust under variation of the initial temperature t_0, the cooling rate r, and the number of neighborhood trials c. For the single best and the best pair of parameter settings these variations led to a change of the average relative deviation of 2 % in average and at most 7.5 %. The quality of the results is more sensitive to the choice of the neighborhood probability α and the number of iterations per temperature level n. Their variation leads to a much slower convergence to good solutions for instances A2-1 and B3, in particular for the high levels.

Table 4 Lower bounds, best known values, initial values, and relative deviation of the best known value from the lower bound

Inst.	*LB*	BKV	C_0	Rel.dev.(‰)
A1-1	44,306,390	44,306,501	49,528,750	0.00
A1-2	777,530,730	**777,532,177**	1,061,649,570	0.00
A1-3	583,005,700	583,005,717	583,662,270	0.00
A1-4	242,387,530	249,742,154	632,499,600	30.34
A1-5	727,578,290	727,578,309	782,189,690	0.00
A2-1	0	167	391,189,190	
A2-2	13,590,090	746,423,338	1,876,768,120	53,924.09
A2-3	521,441,700	1,209,073,257	2,272,487,840	1,318.71
A2-4	1,680,222,380	**1,680,368,578**	3,223,516,130	0.09
A2-5	307,035,180	**307,150,825**	787,355,300	0.38
B-1	3,290,754,940	3,326,577,129	7,644,173,180	10.89
B-2	1,015,153,860	1,015,535,950	5,181,493,830	0.38
B-3	156,631,070	**156,704,281**	6,336,834,660	0.47
B-4	4,677,767,120	4,677,792,539	9,209,576,380	0.01
B-5	922,858,550	**922,944,697**	12,426,813,010	0.09
B-6	9,525,841,820	**9,525,851,483**	12,749,861,240	0.00
B-7	14,833,996,360	14,834,456,201	37,946,901,700	0.03
B-8	1,214,153,440	**1,214,291,143**	14,068,207,250	0.11
B-9	15,885,369,400	15,885,437,256	23,234,641,520	0.00
B-10	18,048,006,980	18,048,187,105	42,220,868,760	0.01
X-1	3,023,565,050	3,044,418,078	7,422,426,760	6.90
X-2	1,001,403,470	1,002,379,317	5,103,634,830	0.97
X-3	0	**69,970**	6,119,933,380	
X-4	4,721,558,880	4,721,591,023	9,207,188,610	0.01
X-5	0	**54,132**	12,369,526,590	
X-6	9,546,930,520	**9,546,936,159**	12,753,566,360	0.00
X-7	14,251,967,330	14,252,476,508	37,763,791,230	0.04
X-8	0	**29,193**	11,611,565,600	
X-9	16,125,494,300	16,125,562,162	23,146,106,380	0.00
X-10	17,815,790,830	17,815,989,054	42,201,640,770	0.01

New best known values found in this study in bold

6.3 Analysis of the lower bounds

In this section we explore results obtained by the proposed lower bound. Table 4 shows the value of the lower bound (*LB*), the best known value (BKV), the initial solution value (C_0), and the relative deviation (in ‰) of the best known value from the lower bound for all instances. Note that for instances which have no balance requirements ($|\mathcal{D}| = 0$ in Table 2), $LB = LB^L$. We have found 11 better upper bounds in our experiments described below, which are highlighted in bold in the table.

The lower bound is generally very close to the best known values, and never deviates more than 3.1 %, except for instances A2-2 and A2-3. Instances with a lower bound of 0, for which

Table 5 Lower bounds from the linear relaxation of the ILP model

Inst.	$\varDelta LC$	Rel.dev.(‰)	Time (s)
A1-1	91	0.00	0.01
A1-2	18	0.00	27.47
A1-3	1	0.00	23.46
A1-4	7,009	30.31	7.67
A1-5	6	0.00	0.26
A2-1	66	–	2.52
A2-2	15,759,126	24,432.48	725.77
A2-3	51,747,796	1,109.38	676.25
A2-4	8,398	0.08	40.47
A2-5	5,481	0.36	28.02

the relative deviation is undefined, usually permit very small objective function values. The good bounds can be explained by the relation of the move costs to the resource costs. For the cost multipliers given in Sect. 6.1 the total move costs never exceed $2001p + 10{,}000$. The total resource costs per process are at least a factor three larger, and in average about a factor of 30 larger. Thus, as long as the safety capacities or the balance requirements are not satisfied, it is usually advantageous to reassign processes, if possible. The small relative deviations show that an almost optimal balance can be achieved in most of the instances.

Table 5 compares the lower bound *LB* to the lower bound obtained by the linear relaxation of the integer program of Sect. 3 (LP lower bound). Results are given only for instance group A, since the models were too large to be solved for the other instance groups within a reasonable time. The table reports the absolute difference between the LP lower bound and *LB* ($\varDelta LB$), the relative deviation of the best known value from the LP lower bound, and the time to compute the LP lower bound.

The LP lower bound is always better than *LB*, but needs considerably more computation time. For most of the instances the improvement is insignificant, and the relative deviations are almost the same. Again, instances A2-2 and A2-3 are the exception. Here the LP lower bound improves most, but the final value still has a relative deviation more than 100 % from the best known value.

6.4 Exact solutions of the mathematical model

Table 6 shows the results found solving the mathematical model of Sect. 3 with CPLEX for instance group A. The results have been obtained running two threads for a time limit of one hour (CPLEX 1h) and ten hours (CPLEX 10h). For both, the table reports for each instance the best value found, the relative deviation from the best known value, and the solution time. The results for ten hours are reported only for the instances which could not be solved in an hour. Instances A1-1, A1-3, and A1-5 could be solved optimally within the limits of the chosen precision (10^{-5}). The remaining instances could not be solved optimally within ten hours, and the final gaps are significantly worse than those obtained by any of the heuristics presented in the next subsection.

6.5 Heuristic results

We ran all the heuristics from Table 3 for 5 min as well as 30 min, to study how solution quality improves for longer computation times. Table 7 gives an overview of the results. For

Table 6 CPLEX results for instance group A

Inst.	CPLEX (1h)			CPLEX (10h)		
	Value	Gap (%)	Time (s)	Value	Gap (%)	Time (s)
A1-1	44,306,501	0.00	25.7	–	–	–
A1-2	777,533,037	0.00	3,360.5	–	–	–
A1-3	583,006,619	0.00	403.4	–	–	–
A1-4	276,332,151	10.65	3,600.0	266,605,122	6.75	36,000
A1-5	727,578,311	0.00	4.6	–	–	–
A2-1	3,861,908	2,312,419.76	3,600.0	215	28.74	36,000
A2-2	1,876,768,120	151.43	3,600.0	805,842,996	7.96	36,000
A2-3	2,272,487,840	87.95	3,600.0	1,315,049,275	8.77	36,000
A2-4	2,366,217,491	40.82	3,600.0	2,052,277,679	22.13	36,000
A2-5	740,879,654	141.21	3,600.0	566,239,468	84.35	36,000

Table 7 Summary of the experimental results

	5 m		30 m		5 m		30 m	
	Score	#	Score	#	Rel.dev.	#	Rel.dev.	#
S41	**124.6**	**13**	**73.8**	8	36.5	**22**	32.2	22
S38	388.3	8	336.7	10	27.0	18	5.8	19
S23	641.5	8	361.2	**17**	**13.4**	**22**	**2.6**	**23**
J25	897.9	2	878.3	5	20,655.3	12	20,577.2	15
J33	1,797.1	5	1,417.4	5	20,571.3	17	11.4	15
S14	1,761.7	1	1,522.6	1	95,038.3	12	94,997.9	12
S34	1,727.7	8	1,595.2	6	90,123.1	13	90,098.8	14

each method, we report the total score obtained within the two time limits as well as the relative deviation (in %) from the best known value. For each measure, we also report the number of instances for which a method was best among all seven (#). The scores (relative distance of the solution) have been computed with respect to the best known values. The best methods are shown in bold.

The scores after 5 min represent the challenge conditions, and our results confirm the ranking of the challenge for the first three methods (S41, S38, S23), but invert it for the last three, which all have very similar scores. The scores after 30 min rank the methods exactly according to the challenge results. All methods, except S38 and J25 improve their result significantly with more computation time available. Our method consistently ranks third in score, obtains the same number of best solutions as S38 after 5 min, and the highest number of best solutions after 30 min.

With respect to relative deviations the difference between the methods is more pronounced. For a time limit of 5 min the first three obtain acceptable results, being within 33 % of the best known value, in average. The gaps of the remaining methods are very large, and after 30 min only J33 is able to close this gap. Our method performs best with respect to this metric, and finds the same number of best solutions as S41 after 5 min, and one more after 30 min.

Table 8 Scores ($\times 10^5$) after 30 min

Inst.	S23	S41	S38	J25	J33	S14	S34
A1-1	0.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
A1-2	**0.1**	0.6	91.1	947.2	34.5	1,999.1	1,954.7
A1-3	0.4	0.1	**0.0**	**0.0**	0.3	0.1	0.1
A1-4	1,037.9	1,521.0	**0.0**	3,728.3	819.1	3,548.1	2,769.3
A1-5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.2	**0.0**
A2-1	**0.0**	**0.0**	**0.0**	263.2	**0.0**	1,153.7	1,153.7
A2-2	3,925.6	**0.0**	2,970.3	6,564.6	12,417.1	11,790.9	11,238.0
A2-3	3,267.5	**0.0**	3,638.0	4,832.3	9,702.0	9,806.7	7,552.6
A2-4	**1.6**	10.3	3.9	24.6	630.1	584.1	1,791.3
A2-5	**12.5**	400.8	2,755.9	8,251.0	15,004.1	12,053.7	18,271.1
B-1	530.9	54.2	**0.0**	863.2	1,843.4	1,351.7	1,462.0
B-2	0.9	**0.0**	19.2	172.1	11.5	28.2	200.0
B-3	**0.0**	4.8	2.1	7.4	1.6	113.2	30.2
B-4	0.6	1.3	**0.0**	0.4	0.9	1.2	0.1
B-5	**0.0**	3.2	0.4	5.5	0.4	344.4	0.5
B-6	**0.0**	0.2	0.1	**0.0**	**0.0**	0.6	**0.0**
B-7	**0.0**	1.6	0.2	14.8	8.9	79.6	7.0
B-8	**0.0**	0.9	0.5	1.4	0.9	88.0	0.6
B-9	0.3	0.5	0.9	1.4	0.4	0.7	**0.0**
B-10	**0.1**	0.7	1.8	1.3	1.0	1.3	0.7
X-1	2,032.5	**203.5**	527.1	498.9	1,898.5	2,188.0	1,272.4
X-2	21.9	**0.0**	88.3	158.9	116.1	48.4	116.8
X-3	**0.1**	4.1	1.0	3.8	1.2	64.9	1.3
X-4	0.5	1.8	**0.0**	0.6	1.2	1.3	0.1
X-5	**0.0**	0.8	**0.0**	0.9	0.1	167.8	0.4
X-6	**0.0**	0.2	**0.0**	**0.0**	**0.0**	0.5	**0.0**
X-7	**0.2**	1.8	0.4	4.0	28.4	1.2	33.1
X-8	**0.0**	0.9	0.1	0.3	0.1	259.5	0.3
X-9	0.5	0.5	1.0	1.5	0.4	0.7	**0.0**
X-10	**0.1**	0.7	0.2	0.9	0.5	1.3	0.2
Averages	361.2	**73.8**	336.7	878.3	1,417.4	1,522.6	1,595.2

Comparing the results for the different time limits we can observe that the first three methods (S41,S38,S23) obtain better scores after 5 min than the remaining methods in 30 min. Thus, for this group of methods, it seems unlikely that their better performance can be explained only based on a more efficient implementation.

Tables 8 and 9 report detailed scores and relative deviations for all 30 instances and all methods. The scores are given in units of 10^{-5}, and have been determined by a single run under challenge conditions. The relative deviations are from the same run, but we additionally report the minimum, average, and maximum relative deviation over 20 replications for our method. For each instance the best value found by any of the methods is shown in bold. The results obtained after 5 min are very similar and thus are not presented. We also provide as

Table 9 Relative deviations after 30 min

Inst.	S23			*S*41	*S*38	J25	J33	S14	S34
	Min	Avg	Max						
A1-1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
A1-2	**0.0**	**0.0**	**0.0**	**0.0**	0.1	1.3	**0.0**	2.7	2.7
A1-3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
A1-4	2.6	4.4	6.2	3.9	**0.0**	9.4	2.1	9.0	7.0
A1-5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
A2-1	**6.0**	11.9	18.0	86.2	12.6	>600 K	76.0	>2.7 M	>2.7 M
A2-2	7.2	11.7	18.7	**0.0**	7.5	16.5	31.2	29.6	28.3
A2-3	4.3	8.2	12.4	**0.0**	6.8	9.1	18.2	18.4	14.2
A2-4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.2	1.1	3.4
A2-5	**0.0**	0.2	0.7	1.0	7.1	21.2	38.5	30.9	46.8
B-1	0.1	1.2	2.1	0.1	**0.0**	2.0	4.2	3.1	3.4
B-2	**0.0**	**0.0**	**0.0**	**0.0**	0.1	0.9	0.1	0.1	1.0
B-3	**0.0**	**0.0**	**0.0**	0.2	0.1	0.3	0.1	4.6	1.2
B-4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
B-5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	**0.0**	4.6	**0.0**
B-6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
B-7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.2	**0.0**
B-8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1.0	**0.0**
B-9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
B-10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
X-1	3.2	4.2	5.5	**0.5**	1.3	1.2	4.6	5.3	3.1
X-2	**0.0**	0.1	0.1	**0.0**	0.4	0.8	0.6	0.2	0.6
X-3	**0.0**	19.3	48.2	358.1	85.8	328.3	105.2	5,679.1	111.6
X-4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
X-5	**0.0**	8.8	23.4	172.4	4.8	194.9	27.6	38,352.8	86.1
X-6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
X-7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	**0.0**	0.1
X-8	**0.0**	9.1	26.0	343.6	47.1	101.8	31.9	> 100K	100.2
X-9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
X-10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
Averages	**0.8**	2.6	5.4	32.2	5.8	20,577.2	11.4	94,997.9	90,098.8

a reference for followup research the average values found by our method after 5 min and 30 min in Tables 10 and 11, respectively.

The results show that the quality of the solutions found by our algorithm is robust. The average relative deviation of 5.4 % of the worst of 20 replications is still slightly better than the relative deviation obtained by the second best method S38. We also have compared our results to the iterated local searches MS-ILS-PP of Masson et al. (2012) and IP-RILS of Lopes et al. (2014). We obtain in both cases better scores and relative deviations in 5 min. Our average values of 20 replications are better than those of MS-ILS-PP of 40 replications

Table 10 Average values of 20 replications found after 5 min

Inst.	A	B	X
1	44,306,985.5	3,459,833,142.7	3,237,614,038.8
2	777,537,895.3	1,015,704,907.8	1,003,788,777.9
3	583,009,352.8	181,348,258.5	179,981.2
4	264,528,485.5	4,677,964,379.8	4,721,796,851.9
5	727,578,312.0	923,173,408.1	110,686.8
6	190.8	9,525,862,876.3	9,546,942,917.9
7	890,702,446.1	14,836,260,664.1	14,258,905,790.5
8	1,359,680,997.5	1,214,495,735.8	46,924.1
9	1,680,528,502.8	15,885,979,673.8	16,126,087,639.2
10	313,041,524.4	18,049,055,327.8	17,816,995,962.0

Table 11 Average values of 20 replications found after 30 min

Inst.	A	B	X
1	44,306,935.0	3,367,756,722.7	3,171,031,536.8
2	777,532,661.6	1,015,563,820.5	1,003,019,443.7
3	583,007,874.2	156,728,583.9	83,498.4
4	260,813,622.2	4,677,835,006.5	4,721,644,542.6
5	727,578,310.5	922,967,506.8	58,879.9
6	186.8	9,525,852,451.2	9,546,937,086.3
7	834,017,083.5	14,834,641,717.0	14,252,628,684.7
8	1,308,179,031.4	1,214,305,764.2	31,839.3
9	1,680,401,671.3	15,885,549,414.0	16,125,679,039.0
10	307,842,073.5	18,048,290,065.4	17,816,108,096.6

in 23 of the 30 instances, and better than those of IP-RILS in 100 replications, in 16 of the 20 instances of groups B and X for which Lopes et al. (2014) provide results.

The detailed results also show that the scores and relative deviations are dominated by a few hard instances, and most instances can be easily solved. We have confirmed this by an additional experiment, in which we have substituted the Metropolis acceptance criterion in our algorithm by a simple criterion, that accepts only better solutions, turning it into a monotone, stochastic local search. In 5 min this method finds solutions with a relative deviation of less than 0.1 % from the best lower bound for half of the instances (namely A1-1, A1-3, A1-5, B4–B10, X4, X6, X7, X9, X10). Five more instances make reasonable progress to the best known values in 30 min. Therefore, these instances are easy to solve, but the Metropolis acceptance criterion often speeds up the solution considerably.

For the scores, instance group A (in particular instances A1-4, A2-2, A2-3, A2-5) is hardest, followed by instances B-1, B-2, X-1, X-2. In comparison with Table 4 it is clear that instances whose best known values are far from the lower bounds tend to be more difficult to solve. Since the lower bound accounts only for load and balance costs, a good lower bound indicates that an almost ideal balance of the resources can be achieved. Conversely, what seems to make the instances difficult is a hard multi-dimensional packing subproblem. All difficult instances, except A1-4, have the highest number of 12 resources, with 4 of them being transient, which restricts the movements of the processes, and a safety capacity of 0 for all resources on 4 % of the machines (called "turndown" by the organizers), which must be emptied to achieve a low load cost.

We finally have conducted experiments to evaluate the contribution of the Metropolis acceptance criterion and neighborhoods to the performance of our heuristic, which are the two most important components. As mentioned above, when the Metropolis acceptance criterion is substituted for an acceptance of only better solutions, 15 instances still are easy to solve in 5 min, but the remaining instances make slow or no progress to better solutions within 30 min, while the Metropolis acceptance criterion speeds up convergence. We have also substituted the Metropolis criterion by a late acceptance criterion (Burke and Bykov 2012). We repeated the calibration for parameters α, c using the same ranges as before, and the late acceptance parameter $k \in \{100, 1{,}000, 10{,}000, 100{,}000\}$. The best pair of parameter settings was $k = 100$, $c = 50$ for the first thread, $k = 10{,}000$, $c = 100$ for the second thread, and $\alpha = 0.5$ for both. Its performance is comparable to that of methods S14 and S34, and we conclude that the Metropolis acceptance criterion contributes significantly to the solutions. The same holds with respect to the second component: when using only the shift neighborhood or the swap neighborhood, the solution quality for some of the hard instances is substantially worse, so both are essential for a good overall solution.

7 Conclusions

We have proposed a mathematical formulation, a lower bound, and a heuristic method based on simulated annealing for the MRP. The method has ranked overall fourth in the 2012 ROADEF/EURO challenge, and our experiments confirm that it generally obtains solutions of good quality, comparable to the best methods of the competition. Our method obtains the best results when considering relative deviations, and often finds the best solution among the methods considered. Our experiments show that these conclusions remain valid for a longer computation time. Compared to other approaches, we believe an additional advantage of our method is its simplicity: its uses two elementary neighborhoods and a simple sampling strategy within a standard simulated annealing heuristic.

References

Brandt, F., Völker, M., & Speck, J. (2012). Constraint-based large neighborhood search for machine reassignment. In *25th European conference on operational research*, Vilnius.

Burke, E. K., & Bykov, Y. (2012). *The late acceptance hill-climbing heuristic*. Tech. Rep. CSM-192, Computing Science and Mathematics, University of Stirling.

Cerny, V. (1985). Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*, 41–51.

Gavranović, H., Buljubasić, M., & Demirović, E. (2012). Variable neighborhood search for google machine reassignment problem. *Eletronic Notes in Discrete Mathematics*, *39*, 209–216. doi:10.1016/j.endm.2012.10.028.

Google Inc. (2011). Google ROADEF/EURO challenge 2011–2012: Machine reassignment. http://challenge.roadef.org/2012/files/problem_definition_v1.pdf, version 1

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research*, *13*, 311–329.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1989). Optimization by simulated annealing. Part I, Graph partitioning. *Operations Research*, *37*, 865–892.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Lopes, R., Morais, V. W., Noronha, T. F., & Souza, V. A. A. (2014). Heuristics and matheuristics for a real-life machine reassignment problem. *International Transactions in Operational Research*, 1–19.

Malitsky, Y., Mehta, D., O'Sullivan, B., Simonis, H. (2013). Tuning parameters of large neighborhood search for the machine reassignment problem. In *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, pp. 176–192.

Masson, R., Vidal, T., Michallet, J., Penna, P. H. V., Petrucci, V., Subramanian, A., et al. (2012). An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications*, *40*, 5266–5275.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, *8*, 3–30.

Mehta, D., OSullivan, B., & Simonis, H. (2012a). Comparing solution methods for the machine reassignment problem. In *Principles and practice of constraint programming*, pp. 782–797.

Mehta, D., O'Sullivan, B., & Simonis, H. (2012b). Team S38 Google ROADEF challenge 2012. In *European conference on operational research*.

Pécot, M. (2012). *Team S34 Google ROADEF challenge 2012*.

Sansottera, A., Ferrucci, L., Sironi, F., & Calcavecchia, N. (2012). *Team J33 Google ROADEF challenge 2012*. Italy: Politecnico di Milano.

Teypaz, N. (2012). *Team S14 Google ROADEF challenge 2012*. France: Probayes SAS.