Comparing Solution Methods for the Machine Reassignment Problem

Deepak Mehta, Barry O'Sullivan, and Helmut Simonis

Cork Constraint Computation Centre, University College Cork, Ireland
{d.mehta|b.osullivan|h.simonis}@4c.ucc.ie

Abstract. The machine reassignment problem is defined by a set of machines and a set of processes. Each machine is associated with a set of resources, e.g. CPU, RAM etc., and each process is associated with a set of required resource values and a currently assigned machine. The task is to reassign the processes to machines while respecting a set of hard constraints in order to improve the usage of the machines, as defined by a complex cost function. We present a natural Constraint Programming (CP) formulation of the problem that uses a set of well-known global constraints. However, this formulation also requires new global constraints. Additionally, the instances are so large that systematic search is not practical especially when the time is limited. We therefore developed a CP formulation of the problem that is especially suited for a large neighborhood search approach (LNS). We also experimented with a mixed-integer programming (MIP) model for LNS. We first compare our formulations on a set of ROADEF'12 problem instances where the number of processes and machines are restricted to 1000 and 100 respectively. Both MIP and CP-based LNS approaches find solutions that are significantly better than the initial ones and compare well to other submitted entries. We further investigate their performances on larger instances where the number of processes and machines can be up to 50000 and 5000 respectively. The results suggest that our CP-based LNS can scale to large instances and is superior in memory use and the quality of solutions that can be found in limited time.

1 Introduction

The management of data centres provides a rich domain for constraint programming, and combinatorial optimisation in general. The *EU Stand-by Initiative* recently published a Code of Conduct for Energy Efficiency in Data Centres.[1] This report stated that in 2007 Western European data centres consumed 56 Tera-Watt Hours (TWh) of power, which is expected to almost double to 104 TWh per year by 2020. A typical optimisation challenge is workload consolidation whereby one tries to keep servers well utilised such that the power costs are minimised. A naïve model for the problem of loading servers to a desired utilisation level for each resource can be regarded as a multi-dimensional bin packing problem [6]. In such a model the servers are represented by bins and the set of resources define the set of dimensions. Many related optimisation problems arise in this domain, such as the adaptive control of virtualised resources [3], and cluster resource management [5, 9]. A MIP approach to dynamically configuring

[1] http://re.jrc.ec.europa.eu/energyefficiency/html/standby_initiative_data_centers.htm

the consolidation of multiple services or applications in a virtualised server cluster has been proposed [4]. That work both focuses on power efficiency, and considers the costs of turning on/off the servers. Power and migration cost-aware application placement has also been considered [8, 7].

Given the growing level of interest from the optimisation community in data centre optimisation and virtualisation, the 2012 ROADEF Challenge is on the topic of machine reassignment, a common task in virtualisation and service configuration on data centres.[2] We present a mixed integer programming and a constraint programming solution for the problem. We also present a large neighbourhood search scheme for both. Our results show that as the problem size grows, the CP approach is significantly more scalable than the MIP approach. This demonstrates the applicability of CP, and in particular LNS, as a promising optimisation approach for this application domain.

2 Problem Description and Notations

In this section, we describe the 2012 machine reassignment problem of ROADEF/EURO Challenge in collaboration with Google. For more details the reader is referred to the specification of the problem, which is available online[3]. We also introduce some notations that will be used throughout the paper. Let M be the set of machines and P be the set of processes. A solution of the machine reassignment problem is an assignment of each process to a machine subject to a set of constraints. The objective is to find a solution that minimizes the cost of the assignment.

2.1 Constraints

Capacity Constraints. Let R be the set of all resources (e.g., CPU, RAM, etc.). Let c_{mr} be the capacity of machine m for resource r and let r_{pr} be the requirement of process p for resource r. The usage by a machine m of resource r, denoted by u_{mr}, is equal to the sum of the amount of resource required by processes which are assigned to machine m. The usage by a machine of a resource should not exceed the capacity of the resource.

Conflict Constraints. A service is a set of processes. Let S be the set of services that partition P. The processes of a service should be assigned to different machines.

Spread Constraints. A location is a set of machines. Let L be the set of all locations that partition the machines. We abuse the notation by using $L(m)$ to denote the location of machine m. The processes of a service s should be assigned to the machines such that their corresponding locations are spread over at least $spreadmin_s$ number of locations.

Neighborhood Constraints. A neighborhood is a set of machines that partition the machines. Let N be the set of all neighborhoods. We abuse the notation by using $N(m)$ to denote the neighborhood of machine m. If service s depends on service s' then it is denoted by $\langle s, s' \rangle$. Let D be the set of such dependencies. If $\langle s, s' \rangle \in D$ then the set of the neighborhoods of the machines assigned to the processes of s must be a subset of the set of the neighborhoods of the machines assigned to the processes of service s'.

[2] http://challenge.roadef.org/2012/en/index.php

[3] http://challenge.roadef.org/2012/files/problem_definition_v1.pdf

Transient Usage Constraints. Let o_p be the original machine assigned to a process p. When a process is moved from o_p to another machine, some resources, e.g., hard disk, are required in both source and target machines during the move. These resources are called transient resources. Let $T \subseteq R$ be the set of transient resources. The transient usage of a machine m for a resource $r \in T$ is equal to sum of the amount of resource required by processes whose original or current machine is m. The transient usage of a machine for a given resource should not exceed the capacity of the resource.

2.2 Costs

Load Cost. It is usually not a good idea to use the full capacity of some resource. The safety capacity limit provides a soft limit, any use above that limit incurs a cost. Let sc_{mr} be the safety capacity of machine m for resource r. The load cost for a resource r is equal to $\sum_{m \in M} \max(0, u_{mr} - sc_{mr})$.

Balance Cost. In order to balance the availability of resources, a balance b can be defined by a triple of resources r_b^1 and r_b^2, and a multiplier t_b. Let B be the set of all such triples. For a given triple b, the balance cost is $\sum_{m \in M} \max(0, t_b \times A(m, r_b^1) - A(m, r_b^2))$ with $A(m, r) = c_{mr} - u_{mr}$.

Process Move Cost. Let pmc_p be the cost of moving a process p from its original machine to any other machine. The process move cost is the sum of all pmc_p such that p is moved from its original machine.

Service Move Cost. To balance the movement of processes among services, a service move cost is defined as the maximum number of moved processes for only services.

Machine Move Cost. Let $mmc(m_1, m_2)$ be the cost of moving a process from a machine m_1 to a machine m_2. Obviously, if m_1 is equal to m_2 then this cost is 0. The machine move cost is the sum of these costs for all processes.

Objective Function. The objective is to minimize the weighted sum of all load-costs, balance-costs, service-move cost, process-move and machine-move cost of each process. Let w_r be the weight of the load-cost for $r \in R$, v_b be the weight of the balance cost for $b \in B$, w^{pmc} be the weight for process move cost, w^{smc} be the weight for service move cost, and w^{mmc} be the weight for machine move cost.

3 Finite Domain Model

In this section, we present a natutal finite domain constraint programming formulation of the machine reassignment problem.

3.1 Variables

For each process p, we use three integer variables: (1) a variable x_p which indicates the machine to which process p is assigned, (2) a variable l_p indicating the location of the machine to which process p is assigned, and (3) a variable n_p indicating the neighborhood of the machine to which process p is assigned.

3.2 Constraints

We use the constraint format of the *Global Constraint Catalog* [1]. Of course, the names and arguments of constraints for specific constraint solvers may differ.

Projection. Let LT and NT be two integer arrays for mapping a machine to its location and its neighborhood. We need element constraints to project from the decision variable x_p to the location and the neighborhood respectively:

$$\forall_{p \in P}: \quad \texttt{element}(x_p, \text{LT}, l_p), \tag{1}$$

$$\forall_{p \in P}: \quad \texttt{element}(x_p, \text{NT}, n_p). \tag{2}$$

Capacity. The hard capacity constraints can be expressed with a `bin_packing_capa` global constraint for each resource:

$$\forall_{r \in R}: \quad \texttt{bin\_packing\_capa}([(m, c_{mr}) | m \in M], [(x_p, r_{pr}) | p \in P]). \tag{3}$$

Conflict. All processes in the same service must be allocated to different machines, i.e. they must be alldifferent:

$$\forall_{s \in S}: \quad \texttt{alldifferent}([x_p | x \in s]). \tag{4}$$

Spread. A service s must run in at least $spreadmin_s$ locations:

$$\forall_{s \in S}: \quad \texttt{atleast\_nvalue}(spreadmin_s, [l_p | p \in s]). \tag{5}$$

Dependency. If a service s depends on another s', then that service must run in all neighborhoods where s is running:

$$\forall \langle s, s' \rangle \in D : \texttt{used\_by}([n_{p'} | p' \in s'], [n_p | p \in s]). \tag{6}$$

Transient. For the transient resource, we have to add some optional tasks, which only occur if the process is not assigned to its original value o_p. Different constraint systems will require different techniques to implement this. In the best case, optional tasks are available, or resource height can be a variable. In the worst case, dummy tasks with escape values should be considered:

$$\begin{aligned} \forall_{r \in T}: \quad & \texttt{bin\_packing\_capa}([(m, c_{mr}) | m \in M], \\ & [(x_p, r_{pr}) | p \in P] \cup [(o_p, r_{pr}) | x_p \neq o_p]). \end{aligned} \tag{7}$$

3.3 Objective

The five different costs described in Section 2 depend on the assignment of a machine to a process. A standard way of modelling the objective function would be an additive, weighted sum of these costs. However, it seems unlikely that such a sum of the different cost factors would allow any significant constraint propagation. Furthermore, systematic search does not seem to be a feasible approach especially when the size of the instance is very large and the solution-time is very limited. Therefore it did not lead to a working system.

4 A Mixed Integer Linear Programming Model

In this section we present a mixed integer linear programming formulation of the machine reassignment problem. We use the notation $a := b$ to abbreviate expression b with name a. We may, but do not have to, create a new variable for a.

4.1 Variables

x_{pm} A binary variable that indicates if process p is running on machine m. Thus, x_{po_p} indicates that the process is not moving, while $(1 - x_{po_p})$ indicates that the process is moving from its original assignment o_p.

y_{sl} A binary variable that indicates if service s is running in location l.

z_{sn} A binary variable that states that service s is running in neighborhood n.

lc_{mr} A continuous (non-negative) variable for resource r on machine m that shows how much the usage exceeds the safety capacity.

bc_{bm} A continuous (non-negative) variable for balance b on machine m that links resources r_b^1 and r_b^2 with multiplier t_b.

smc A continuous (non-negative) variable for cost of moving processes of the services.

The following may be used as abbreviations or variables depending on the system that is used: (1) u_{mr} denotes resource usage of machine m for resource r, (2) a_{mr} denotes free capacity of machine m for resource r, (3) lc_r denotes load cost of resource r, (4) bc_b denotes balance cost of balance b, (5) pmc denotes process move cost, (6) smc_s contribution of service s to service move cost, (7) mmc_p contribution of process p to machine move cost, and (8) mmc denotes machine move cost.

4.2 Constraints

Assignment. Every process must be allocated to exactly one machine.

$$\forall_{p \in P} : \quad \sum_{m \in M} x_{pm} = 1 \tag{8}$$

Capacity. For each resource, the resource use for all processes on one machine must be below the resource limit of that machine. We denote with u_{mr} the resource use on machine m for resource r by all process which run on this machine.

$$\forall_{r \in R} \forall_{m \in M} : \quad u_{mr} := \sum_{p \in P} r_{pr} x_{pm} \leq c_{mr} \tag{9}$$

Conflict. Processes belonging to the same service can not be run on the same machine.

$$\forall_{s \in S} \forall_{m \in M} : \quad \sum_{p \in s} x_{pm} \leq 1 \tag{10}$$

Spread. A service must run in at least spreadmin locations. We introduce y_{sl} to denote if one of the processes in service s is running on one of the machines in location l.

$$\forall_{l \in L} \forall_{s \in S} : \quad \sum_{m \in l} \sum_{p \in s} x_{pm} \leq |l||s| y_{sl} \tag{11}$$

$$\forall_{l \in L} \forall_{s \in S} : \quad \sum_{m \in l} \sum_{p \in s} x_{pm} \geq y_{sl} \tag{12}$$

$$\forall_{s \in S} : \quad \sum_{l \in L} y_{sl} \geq \text{spreadmin}_s \tag{13}$$

Dependency (I). If a service depends on another service, then that service must run in the same neighborhood.

$$\forall_{\langle s,s' \rangle \in D} \forall_{n \in N} \forall_{p \in s} \forall_{m \in n} : \quad x_{pm} \leq \sum_{p' \in s'} \sum_{m' \in n} x_{p'm'} \tag{14}$$

Dependency (II). Dependency (I) seems to require too many constraints, we can introduce binary variables z_{sn} which indicate that service s is run in neighborhood n. We then have the constraints to link the x_{pm} and the z_{sn} variables:

$$\forall_{n \in N} \forall_{s \in S} : \quad \sum_{m \in n} \sum_{p \in s} x_{pm} \leq |n||s| z_{sn} \tag{15}$$

$$\forall_{n \in N} \forall_{s \in S} : \quad \sum_{m \in n} \sum_{p \in s} x_{pm} \geq z_{sn} \tag{16}$$

For every service dependency and every neighborhood, we have an implication:

$$\forall_{\langle s,s' \rangle \in D} \forall_{n \in N} : \quad z_{sn} \leq z_{s'n} \tag{17}$$

Dependency (III). We can reduce the number of constraints also by reorganizing the first version. On the left hand side, we can add all x_{pm} variables for the same process, as their sum is always less than or equal to 1. On the right rand side, we can reuse the sum for multiple left hand sides, so don't have to create these sums over and over again. This seems to make approach (III) feasible for first set of instances of ROADEF.

$$\forall_{\langle s,s' \rangle \in D} \forall_{n \in N} \forall_{p \in s} : \quad \sum_{m \in n} x_{pm} \leq \sum_{p' \in s'} \sum_{m' \in n} x_{p'm'} \tag{18}$$

In our implementation, the third alternative for the dependency constraint is used.

Transient. For transient resources, the resource use must include the original and the new machine, not just the new machine. It is important to avoid double counting if the process does not move.

$$\forall_{r \in T} \forall_{m \in M} : \quad \sum_{p \in P, o_p = m} r_{pr} + \sum_{p \in P, o_p \neq m} r_{pr} x_{pm} \leq c_{mr} \tag{19}$$

Note that the transient resource type only affects this capacity constraint, not how the load cost is calculated.

4.3 Objective

Load Cost. We need continuous, non-negative variables lc_{mr} to express this cost. Recall that u_{mr} denotes the resource use of resource r on machine m, and integer sc_{mr} is the safety capacity limit for resource r on machine m.

$$\forall_{r \in R} \forall_{m \in M} : \quad lc_{mr} \geq u_{mr} - sc_{mr} \tag{20}$$

$$\forall_{r \in R} : \quad lc_r := \sum_{m \in M} lc_{mr} \tag{21}$$

Balance Cost. We define the free capacity for resource r on machine m as

$$\forall_{m \in M} \forall_{r \in R} : \quad a_{mr} := c_{mr} - u_{mr} \tag{22}$$

We express the contribution of balance cost b on machine m as

$$\forall_{b \in B} \forall_{m \in M} : \quad bc_{bm} \geq t_b \times a_{mr_b^1} - a_{mr_b^2} \tag{23}$$

The total contribution of balance cost b is summed over all machines as follows:

$$\forall_{b \in B} : \quad bc_b := \sum_{m \in M} bc_{bm} \tag{24}$$

Process Move Cost. This considers if a process is moved from its original assignment. Each process p can be weighted individually with constants pmc_p.

$$pmc := \sum_{p \in P} (1 - x_{po_p}) pmc_p \tag{25}$$

Service Move Cost. We require a continuous, non-negative variable smc to express this cost:

$$\forall_{s \in S} : \quad smc_s := \sum_{p \in s} (1 - x_{po_p}) \tag{26}$$

$$\forall_{s \in S} : \quad smc \geq smc_s \tag{27}$$

smc_s denotes how many processes in service s are assigned to new machines, smc is bounded by those limits. This requires smc to be used inside the cost function for minimization with a non-negative coefficient.

Machine Move Cost. This considers the cost matrix $mmc()$ for moving processes between machines. This gives different weights for from-to machine pairs, but does not distinguish processes.

$$\forall_{p \in P} : \quad mmc_p := \sum_{m \in M} x_{pm} mmc(o_p, m) \tag{28}$$

$$mmc := \sum_{p \in P} mmc_p \tag{29}$$

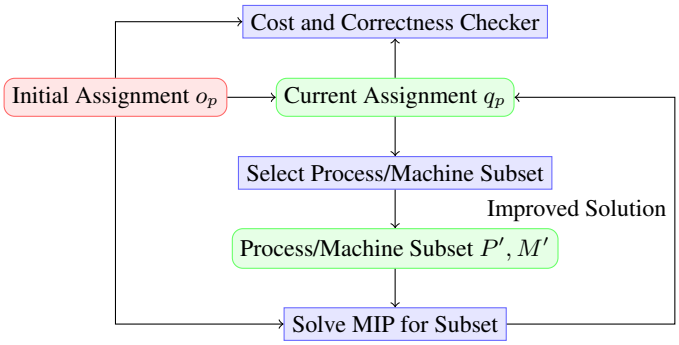

Fig. 1. Principles of the LNS-based approach

Overall Cost. The overall cost is a weighted sum of all the different cost elements defined above. The weight factors are defined as part of the input data of the problem instance. The objective is to minimize the following:

$$Cost = \sum_{r \in R} w_r lc_r + \sum_{b \in B} v_b bc_b + w^{pmc} pmc + w^{smc} smc + w^{mmc} mmc \qquad (30)$$

4.4 Solution Method

We use a large neighborhood search starting from the MIP formulation of the problem. The overall solution method is shown in Figure 1. We maintain a current assignment, which is initialized by the initial solution given as input. At every iteration step, we select a subset of the processes to be reassigned, and setup the MIP model, freezing all other processes. We solve the resulting, small MIP with a timeout, and keep the best solution found as our new current assignment. At each step, we check the correctness and cost of our current solution by a separate checker to avoid isssues with accuracy and tolerances in the MIP solver.

A key observation was that selecting all processes from only some machines for reassignment works better than selecting only a few processes from many machines. Our subproblem selection therefore is based on selecting a subset of the machines, and allowing all processes on those machines to be reassigned. A set of machines are selected by solving a small combinatorial optimization problem. We compare the current usage, denoted by cu_{mr}, of the selected machines (when the Boolean variable $u_m = 1$) with an ideal solution where we can distribute the utilization over all machines such that the area above the safety capacity is minimal. The objective is to maximize the following:

$$\sum_{m \in M} (\sum_{r \in R} \max(0, cu_{mr} - sc_{mr}))u_m - \sum_{r \in R} \max(0, \sum_{m \in M} (cu_{mr} - sc_{mr})u_m)$$

This is optimistic, as we normally will not be able to achieve this totally balanced spread, but this will guide the subset selection to find promising machine subsets. For

some sub-problems this works perfectly. At every step the promised improvement is nearly fully realized by the subsequent MIP run, until the promise (and cost) converge to 0. For problems with significant transient resource elements, the approach works less well, but is still much better than a random selection. We also use a tabu list and a progressive parameter weighting to improve results over all problem instances.

5 CP Model for LNS

We now present a CP model of the problem that facilitates incrementality in the large neighbourhood search method. This helps in creating subproblems efficiently as explained later.

5.1 Variables

Let x_p be an integer variable that indicates which machine is assigned to process p. The domain of x_p is $[0, |M| - 1]$. Let u_{mr} be an integer variable that denotes the usage of resource r on machine m. The domain of u_{mr} is $[0, c_{mr}]$. Let t_{mr} be an integer variable that denotes the transient usage of machine m for transient-resource r. The domain of t_{mr} is $[0, c_{mr}]$. The following may be used as abbreviations or variables depending on the system that is used: (1) um_s denotes a set of machines assigned to the processes of service s, (2) yu_{sl} denotes the number of machines belonging to location l that are assigned to the processes of the service s, (3) nul_s denotes the number of locations used by the processes of the service s, (4) zu_{sn} denotes the number of machines of neighborhood n that are used by service s, (5) zm_{sn} denotes the number of services that enforce that n should be a neighborhood of s, (6) nmp_s denotes how many processes in service s are assigned to new machines, and (7) mmp denotes the maximum number of processes of services that are moved to new machines.

5.2 Constraints

Load Constraints. The usage of resource $r \in R$ on machine m is the sum of the resources required by those processes p that are assigned to machine m:

$$\forall m \in M, r \in R : u_{mr} = \sum_{\forall p \in P \wedge x_p = m} r_{pr}. \tag{31}$$

Transient Constraints. The transient-usage of transient-resource $r \in T$ on machine m is the sum of u_{mr} and the resources required by those processes p whose original machine is m but whose current machine is not m.

$$\forall m \in M, r \in T : t_{mr} = u_{mr} + \sum_{o_p = m \wedge x_p \neq o_p} r_{pr}. \tag{32}$$

Conflict Constraints. The set of machines used by the processes of service s is:

$$um_s := \{x_p : p \in s\}.$$

Processes belonging to the same service s cannot be run on the same machine:

$$\forall s \in S, |s| = |um_s|. \tag{33}$$

Spread Constraints. The number of machines at a location l used by a service s is:

$$\forall s \in S, l \in L, yu_{sl} := |\{x_p | p \in s \wedge L(x_p) = l\}|.$$

The number of locations used by the processes of a service s is:

$$nul_s := |\{l | l \in L \wedge yu_{sl} > 0\}|.$$

The number of locations used by service s should be greater than or equal to the minimum spread of service s:

$$nul_s \geq spreadmin_s. \tag{34}$$

Neighborhood Constraints. The number of machines used by a service s in a neighborhood n is:

$$zu_{sn} := |\{x_p | p \in s \wedge N(x_p) = n\}|.$$

The number of services that want n to be a (mandatory) neighborhood of service s is:

$$zm_{sn} := |\{s' | \langle s', s\rangle \in D \wedge zu_{s'n} > 0\}|.$$

The number of mandatory neighborhoods of service s that are not used by service s is:

$$nmn_s := |\{n \in N \wedge zm_{sn} > 0 \wedge zu_{sn} = 0\}|$$

Each mandatory neighborhood of service s should be used by service s:

$$nmn_s = 0. \tag{35}$$

5.3 Objective

We define the cost of a machine as the sum of the weighted load costs of all resources and the sum of all the weighted balance costs of all the balances:

$$cost_m := \sum_{r \in R} \max(0, u_{mr} - sc_{mr}) \cdot w_r + \sum_{b \in B} \max(0, t_b \cdot a_{mr_b^1} - a_{mr_b^2}) \cdot v_b. \tag{36}$$

We define the cost of a process as the sum of the weighted process move cost and weighted machine move cost:

$$cost_p := \min(1, |x_p - o_p|) \cdot pmc_p \cdot w^{pmc} + mmc(o_p, x_p) \cdot w^{mmc}. \tag{37}$$

We define the cost of a service as the weighted sum of the number of moved processes of the service:

$$cost_s := \sum_{p \in s \wedge x_p \neq o_p} w^{sms}. \tag{38}$$

The overall cost, to be minimized, is:

$$Cost = \sum_{m \in M} cost_m + \sum_{p \in P} cost_p + \max_{s \in S}(cost_s). \tag{39}$$

5.4 Solution Method

We use large neighborhood search for the CP model described above. Using the LNS of MIP for the CP model is a non-starter. Therefore we use a different LNS. In the following we describe how a subproblem is selected and created, and how it is solved.

Subproblem Selection. We first select a number of machines, denoted by k_m, whose processes we want to reassign and then for each selected machine we select a number of processes, bounded by k_p, which we want to reassign. Both k_m and k_p are non-zero positive integers. Initially k_m is set to 1, it is incremented as search progresses, and it is re-initialized to 1 when it exceeds 10. We also compute an ordering on the machines based on Equation (36) after regular intervals. We progressively select one machine from this pre-computed ordering and select the remaining $k_m - 1$ machines randomly.

Depending on the value of k_m a fixed value of k_p is used. If all the processes of a machine are selected then many of them may select the same machine again. Therefore, we restrict that the number of processes selected for a given machine is less than or equal to the half of the average number of processes on a machine. We further restrict that the maximum value of k_p is 10. The total number of processes selected for reassignment is bounded by $k_m \times k_p \leq 40$. Hence, the number of processes selected from a given machine is determined by $\min(40/k_m, \min(|P|/(2 \cdot |M|), 10))$.

Subproblem Creation. For solving large sized problems using LNS one may need to select, create and solve many subproblems. When the time for solving the problem is limited, one challenge is to create a subproblem efficiently. One approach is to create the full problem in memory and reinitialize/recompute the domains of the required variables at each iteration. Depending on the model and the solution technique it may not always be possible to allocate the required memory for this approach, especially considering that the number of processes could be up to 50,000 and the number of machines could be up to 5,000. Another approach could be to allocate the memory and create a subproblem and reinitialize/recompute the domains of the required variables at each iteration, which may not be efficient in terms of time. Creating a subproblem in the context of machine-reassignment problem can be seen as unassigning a set of machines from a set of processes and updating the domains accordingly. We remark that removing values from the domains due to constraint-propagation after an assignment is generally done efficiently in constraint-solvers. However restoring values to the domains after undoing an assignment may not always be straightforward, when no assumption is made on the ordering of assignments. The presented CP-based LNS approach facilitates that domains can be updated efficiently for creating a subproblem.

The two main operations for CP-based LNS for solving machine-reassignment problem are: removing a process from a machine (unassigning a machine from a process), and adding a process to a machine (assigning a machine to a process). When a process p is removed from machine x_p, the domains are updated as shown in Algorithm 1. x_p is removed from the set of machines used by service s (Line 2). The number of machines of location l used by s is decremented (Line 3) and if that becomes 0 (Line 4) the number of locations used by s is also decremented (Line 5). If the number of machines of

Algorithm 1 removeProcessFromMachine(p)

1: $m \leftarrow x_p$; $s \leftarrow S_p$; $n \leftarrow N_m$; $l \leftarrow L_m$
2: $um_s \leftarrow um_s - \{m\}$;
3: $yu_{sl} \leftarrow yu_{sl} - 1$
4: **if** $yu_{sl} = 0$ **then**
5: $\quad nul_s \leftarrow nul_s - 1$
6: $zu_{sn} \leftarrow zu_{sn} - 1$
7: **if** $zu_{sn} = 0$ **then**
8: $\quad$ **if** $zm_{sn} > 0$ **then**
9: $\quad\quad nmn_s \leftarrow nmn_s + 1$
10: $\quad$ **for all** $\langle s, s' \rangle \in D$ **do**
11: $\quad\quad zm_{s'n} \leftarrow zm_{s'n} - 1$
12: $\quad\quad$ **if** $zu_{s'n} = 0 \wedge zm_{s'n} = 0$ **then**
13: $\quad\quad\quad nmn_{s'} \leftarrow nmn_{s'} - 1$
14: $Cost \leftarrow Cost - \sum_{r \in R} \max(0, u_{mr} - sc_{mr}) \times w_r$
15: $Cost \leftarrow Cost - \sum_{b \in B} \max(0, t_b \times (c_{mr_b^1} - u_{mr_b^1}) - (c_{mr_b^2} - u_{mr_b^2})) \times v_b$
16: **for all** $r \in R$ **do**
17: $\quad u_{mr} \leftarrow u_{mr} - r_{pr}$
18: $\quad$ **if** $r \in T$ and $m \neq o_p$ **then**
19: $\quad\quad t_{mr} \leftarrow t_{mr} - r_{pr}$
20: $Cost \leftarrow Cost + \sum_{r \in R} \max(0, u_{mr} - sc_{mr}) \times w_r$
21: $Cost \leftarrow Cost + \sum_{b \in B} \max(0, t_b \times (c_{mr_b^1} - u_{mr_b^1}) - (c_{mr_b^2} - u_{mr_b^2})) \times v_b$
22: **if** $x_p \neq o_p$ **then**
23: $\quad Cost \leftarrow Cost - pmc_p \times w^{pmc}$
24: $\quad nmp_s \leftarrow nmp_s - 1$
25: $\quad$ **if** $mmp = nmp_s + 1$ **then**
26: $\quad\quad mmp \leftarrow \max_{s \in S} nmp_s$
27: $\quad\quad$ **if** $nmp_s + 1 \neq mmp$ **then**
28: $\quad\quad\quad Cost \leftarrow Cost - w^{smc}$
29: $Cost \leftarrow Cost - mmc(o_p, x_p) \times w^{mmc}$

neighborhood n used by s becomes 0 (Line 6), and if the number of services that want n to be included is greater than 0, then the number of mandatory neighborhoods of s is incremented. If n is no longer used by s then the the number of services that want n to be included is decremented for each service s', on which s depends on. For each s' if n is not used anymore then the number of mandatory neighborhoods of s' is also decremented. Lines 14–29 update the cost. First the load and balance costs for all resources and balances are subtracted from the cost (Lines 14–15). Then, the usage and transient usage are updated (Lines 16–19). The load and balance costs are recomputed and added to the cost (Lines 20–21). The process and service move costs are subtracted (Lines 22-28) followed by the subtraction of machine-move cost (Line 29).

Re-optimizing a Subproblem We use systematic search for solving a given subproblem. However, the search is stopped when the number of failures exceed a given threshold. Three important components of a CP-based systematic search are: variable ordering heuristics, value ordering heuristics, and filtering rules. The variable ordering heuristic we use is based on an aggregation of the following information that we maintain for each process p: (a) minimum increment in the objective cost when assigning a machine to process p, (b) total weighted requirement of a process which is the sum of the weighted requirements of all resources, and (c) the number of machines available for p. The value ordering ordering heuristic that is used to select a machine for a given process is based on the minimum cost while ties are broken randomly.

Algorithm 2 addProcessToMachine(p, m)

$s \leftarrow S_p; n \leftarrow N_m; l \leftarrow L_m$
$x_p \leftarrow m$
$Cost \leftarrow Cost + mmc(o_p, x_p) \times w^{mmc}$
if $x_p \neq o_p$ **then**
    $Cost \leftarrow Cost + pmc_p \times w^{pmc}$
    $nmp_s \leftarrow nmp_s + 1$
    if $mmp < nmp_s$ **then**
        $mmp \leftarrow nmp_s$
        $Cost \leftarrow Cost + w^{smc}$
$Cost \leftarrow Cost - \sum_{r \in R} \max(0, u_{mr} - sc_{mr}) \times w_r$
$Cost \leftarrow Cost - \sum_{b \in B} \max(0, t_b \times (c_{mr_b^1} - u_{mr_b^1}) - (c_{mr_b^2} - u_{mr_b^2})) \times v_b$
for all $r \in R$ **do**
    $u_{mr} \leftarrow u_{mr} + r_{pr}$
    if $r \in T$ and $x_p \neq o_p$ **then**
        $t_{mr} \leftarrow t_{mr} + r_{pr}$
$Cost \leftarrow Cost + \sum_{r \in R} \max(0, u_{mr} - sc_{mr}) \times w_r$
$Cost \leftarrow Cost + \sum_{b \in B} \max(0, t_b \times (c_{mr_b^1} - u_{mr_b^1}) - (c_{mr_b^2} - u_{mr_b^2})) \times v_b$
if $zu_{sn} = 0$ **then**
    if $zm_{sn} > 0$ **then**
        $nmn_s \leftarrow nmn_s - 1$
    for all $\langle s, s' \rangle \in D$ **do**
        if $zu_{s'n} = 0 \wedge zm_{s'n} = 0$ **then**
            $nmn_{s'} \leftarrow nmn_{s'} + 1$
        $zm_{s'n} \leftarrow zm_{s'n} + 1$
$zu_{sn} \leftarrow zu_{sn} + 1$
if $yu_{sl} = 0$ **then**
    $nul_s \leftarrow nul_s + 1$
$yu_{sl} \leftarrow yu_{sl} + 1$
$um_s \leftarrow um_s \cup \{m\}$;

At each node of the search tree constraint propagation is performed to reduce the search space. Whenever a machine m is assigned to a process p, the domains are filtered as shown in Algorithm 2, and the affected constraints are checked. Additionally, during subproblem solving, usage and cost based filtering is also applied for removing machines from the domains of the processes. We omit the details of these pruning rules due to lack of space. Let Q be a set of unassigned processes. The minimum load-cost that will be incurred as a result of assigning machines to processes in Q is denoted by lc^{bound}. The minimum balance-cost that will be incurred as a result of assigning machines to the processes in Q is denoted by bc^{bound}. We use the following to compute lc^{bound} and bc^{bound}, during search:

$$lc^{bound} = \sum_{r \in R} \left(\sum_{p \in Q} r_{pr} - \sum_{m \in M} \max(0, sc_{mr} - u_{mr}) \right) \times w_r.$$

$$bc^{bound} = \sum_{b \in B} \left(-t_b \times \sum_{p \in Q} r_{pr_b^1} + \sum_{p \in Q} r_{pr_b^2} \right) \times v_b.$$

The value of lc^{bound} is obtained by adding the total demand for each resource and comparing it to the total safety capacity for that resource on all machines, and multiplying with the appropriate cost-factor. Similarly, the value of bc^{bound} is obtained by considering the weighted difference of total resources and multiplying with the appropriate cost factor. The lower-bound of the objective function is obtained by adding lc^{bound} and bc^{bound} to the current cost of the partial solution.

Table 1. Properties of Instances in Datasets A and B

Set	No.	$\|P\|$	$\|R\|$	$\|T\|$	$\|M\|$	$\|S\|$	$\|L\|$	$\|N\|$	$\|D\|$	$\|B\|$	w_r	v_b	w^{pmc}	w^{smc}	w^{mmc}	mmc^{max}
a1	1	100	2	0	4	79	4	1	0	1	10	10	1	10	100	2
a1	2	1000	4	1	100	980	4	2	40	0	10	-	1	10	100	2
a1	3	1000	3	1	100	216	25	5	342	0	10	-	1	10	100	2
a1	4	1000	3	1	50	142	50	50	297	1	10	10	1	10	100	2
a1	5	1000	4	1	12	981	4	2	32	1	10	10	1	10	100	2
a2	1	1000	3	0	100	1000	1	1	0	0	10	-	1	10	100	0
a2	2	1000	12	4	100	170	25	5	0	0	10	-	1	10	100	2
a2	3	1000	12	4	100	129	25	5	577	0	10	-	1	10	100	2
a2	4	1000	12	0	50	180	25	5	397	1	10	10	1	10	100	2
a2	5	1000	12	0	50	153	25	5	506	0	10	-	1	10	100	2
b	1	5000	12	4	100	2512	10	5	4412	0	10	-	1	10	100	2
b	2	5000	12	0	100	2462	10	5	3617	1	10	10	1	10	100	2
b	3	20000	6	2	100	15025	25	5	16560	0	10	-	1	10	100	2
b	4	20000	6	0	500	1732	50	5	40485	1	10	10	1	10	100	2
b	5	40000	6	2	100	35092	10	5	14515	0	10	-	1	10	100	2
b	6	40000	6	0	200	14680	50	5	42081	1	10	10	1	10	100	0
b	7	40000	6	0	4000	15050	50	5	43873	1	10	10	1	10	100	2
b	8	50000	3	1	100	45030	10	5	15145	0	10	-	1	10	100	2
b	9	50000	3	0	1000	4609	100	5	4337	1	10	10	1	10	100	2
b	10	50000	3	0	5000	4896	100	5	47260	1	10	10	1	10	100	2

6 Experimental Results

In this section we present some results to demonstrate the effectiveness of our approaches, and in particular the scalability of the CP approach. We experimented with the instances of Datasets A and B of EURO/ROADEF'12 Challenge. Table 1 shows the properties of these instances. In set A (B) the maximum number of processes is 1,000 (50,000) and the maximum number of machines is 100 (5,000). For MIP we used CPLEX and the algorithms were implemented in Java. For CP we did not use any existing local search solver, e.g., COMET [2], in order to get more freedom in controlling different aspects of LNS approach. All the algorithms were implemented in C.

Table 2 shows results for dataset A. For each problem instance, we give the initial cost, the best lower bound found (LB), the best solution found by any solver submitted to the ROADEF competition (Best ROADEF) and the best result obtained within 300s for one overall set of parameter settings for our MIP and CP approaches. The full sized MIP problem could not be solved for all the instances of set A. However, both MIP

Table 2. Cost Results for set A obtained within 300 seconds

Problem	Initial	LB	Best ROADEF	MIP-LNS	CP-LNS
a1-1	49,528,750	44,306,501.00	44,306,501	**44,306,501**	**44,306,501**
a1-2	1,061,649,570	777,531,000.00	777,532,896	792,813,766	**778,654,204**
a1-3	583,662,270	583,005,717.00	583,005,717	583,006,527	**583,005,829**
a1-4	632,499,600	242,397,000.00	252,728,589	**258,135,474**	**251,189,168**
a1-5	782,189,690	727,578,309.00	727,578,309	**727,578,310**	727,578,311
a2-1	391,189,190	103.87	198	273	**196**
a2-2	1,876,768,120	526,244,000.00	816,523,983	836,063,347	**803,092,387**
a2-3	2,272,487,840	1,025,730,000.00	1,306,868,761	1,393,648,719	**1,302,235,463**
a2-4	3,223,516,130	1,680,230,000.00	1,681,353,943	1,725,846,815	**1,683,530,845**
a2-5	787,355,300	307,041,000.00	336,170,182	359,546,818	**331,901,091**

Table 3. Cost Results for set B obtained within 300 seconds for CP approach

Name	Initial	LB	CP-LNS	Iterations
b-1	7,644,173,180	3,290,754,940	3,337,329,571	813,519
b-2	5,181,493,830	1,015,153,860	1,022,043,596	477,375
b-3	6,336,834,660	156,631,070	157,273,705	1,271,094
b-4	9,209,576,380	4,677,767,120	4,677,817,475	226,561
b-5	12,426,813,010	922,858,550	923,335,604	968,840
b-6	12,749,861,240	9,525,841,820	9,525,867,169	618,878
b-7	37,946,901,700	14,833,996,360	14,838,521,000	36,886
b-8	14,068,207,250	1,214,153,440,	1,214,524,845	1,044,842
b-9	23,234,641,520	15,885,369,400	15,885,734,072	115,054
b-10	42,220,868,760	18,048,006,980	18,049,556,324	31,688

and CP-based LNS approaches performed well. The approach that outperforms another approach in terms of cost is made bold for each instance of set A. Overall, CP-based LNS outperforms MIP-based LNS.

Table 3 presents results for dataset B. For each instance, we give the initial cost, the lower bound (LB) and the result obtained within 300s for the CP approach, and the number of iterations (Iterations). The results for MIP-based LNS are not presented. The reason is that in each iteration 10 machines are selected, which requires roughly 10 seconds to find an improving solution. This is feasible with 100 machines, but with 5000 not enough combinations are explored in 300 seconds. The results suggest that our CP-based LNS can scale to very large problem instances and is superior both in memory use and the quality of solutions that can be found in limited time. This is mainly because of three factors. First, *selecting a subproblem* by selecting a set of processes from only a few selected machines works better than selecting a few processes from many machines. Second, *creating a subproblem* efficiently by removing a set of processes from their corresponding machines and updating the domains incrementally increases the number of iterations resulted in a greater level of exploration of the search space. Finally, *solving a subproblem* by incrementally maintaining the domains, using cheap constraint checks and using a heuristic that selects a machine that incurs the lowest cost for a given process resulted in efficient exploration of the search-space of the subproblem.

7 Conclusions and Future Work

We presented MIP and CP-based LNS approaches for solving the machine reassignment problem. Empirical results shows that our CP-based LNS approach is scalable, thus suited for solving large instances, and has better anytime-behaviour which is important when solutions must be reported subject to a time limit. The incrementality aspect of the CP-approach allows LNS to create and solve subproblems efficiently, which is a key-factor in finding good quality solutions in a limited time. Currently, we are not taking advantage of any multi-cores that might be available while solving the problem, which we plan to exploit in the future.

Acknowledgments

This work is supported by Science Foundation Ireland Grant No. 10/IN.1/I3032.

References

1. N. Beldiceanu, M. Carlsson, and J. Rampon. Global constraint catalog, 2nd edition (revision a). Technical Report T2012:03, SICS, 2012.
2. Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT Press, 2009.
3. Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 289–302, New York, NY, USA, 2007. ACM.
4. Vinicius Petrucci, Orlando Loques, and Daniel Mosse. A dynamic conguration model for power-efficient virtualized server clusters. In *Proceedings of the 11th Brazilian Workshop on Real-Time and Embedded Systems*, 2009.
5. Kai Shen, Hong Tang, Tao Yang, and Lingkun Chu. Integrated resource management for cluster-based internet services. *SIGOPS Oper. Syst. Rev.*, 36(SI):225–238, 2002.
6. Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of HotPower*, 2008.
7. Malgorzata Steinder, Ian Whalley, James E. Hanson, and Jeffrey O. Kephart. Coordinated management of power usage and runtime performance. In *NOMS*, pages 387–394. IEEE, 2008.
8. Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: Power and migration cost aware application placement in virtualized systems. In Valérie Issarny and Richard E. Schantz, editors, *Middleware*, volume 5346 of *Lecture Notes in Computer Science*, pages 243–264. Springer, 2008.
9. Xiaorui Wang and Ming Chen. Cluster-level feedback power control for performance optimization. In *HPCA*, pages 101–110. IEEE Computer Society, 2008.