Tuning Parameters of Large Neighborhood Search for the Machine Reassignment Problem*

Yuri Malitsky, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis

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

Abstract. Data centers are a critical and ubiquitous resource for providing infrastructure for banking, Internet and electronic commerce. One way of managing data centers efficiently is to minimize a cost function that takes into account the load of the machines, the balance among a set of available resources of the machines, and the costs of moving processes while respecting a set of constraints. This problem is called the *machine reassignment problem*. An instance of this online problem can have several tens of thousands of processes. Therefore, the challenge is to solve a very large sized instance in a very limited time. In this paper, we describe a constraint programming-based Large Neighborhood Search (LNS) approach for solving this problem. The values of the parameters of the LNS can have a significant impact on the performance of LNS when solving an instance. We, therefore, employ the Instance Specific Algorithm Configuration (ISAC) methodology, where a clustering of the instances is maintained in an offline phase and the parameters of the LNS are automatically tuned for each cluster. When a new instance arrives, the values of the parameters of the closest cluster are used for solving the instance in the online phase. Results confirm that our CP-based LNS approach, with high quality parameter settings, finds good quality solutions for very large sized instances in very limited time. Our results also significantly outperform the hand-tuned settings of the parameters selected by a human expert which were used in the runner-up entry in the 2012 EURO/ROADEF Challenge.

1 Introduction

Data centers are a critical and ubiquitous resource for providing infrastructure for banking, Internet and electronic commerce. They use enormous amounts of electricity, and this demand is expected to increase in the future. For example, a report by the *EU Stand-by Initiative* stated that in 2007 Western European data centers consumed 56 Tera-Watt Hours (TWh) of power, which is expected to almost double to 104 TWh per year by 2020.[1] A typical optimization challenge in the domain of data centres is to consolidate machine workload to ensure that

* This work is supported by Science Foundation Ireland Grant No. 10/IN.1/I3032 and by the European Union under FET grant ICON (project number 284715).

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

machines are well utilized so that energy costs can be reduced. In general, the management of data centers provides a rich domain for constraint programming, and combinatorial optimization [13, 16–18].

Context. Given the growing level of interest from the optimization community in data center optimization and virtualization, the 2012 ROADEF Challenge was focused on machine reassignment, a common task in virtualization and service configuration on data centers.[2] Informally, the machine reassignment problem is defined by a set of machines and a set of processes. Each machine is associated with a set of available resources, e.g. CPU, RAM etc., and each process is associated with a set of required resource values and a currently assigned machine. There are several reasons for reassigning one or more processes from their current machines to different machines. For example, if the load of the machine is high, then one might want to move some of the processes from that machine to other machines. Similarly, if the machine is about to shut down for maintenance then one might want to move all processes from the machine. Also, if there is a different location where the electricity price is cheaper then one might want to reassign some processes to the machines at that location such that the total cost of electricity consumption is reduced. In general, 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.

Contributions of this Paper. The machine reassignment problem is one that needs to be solved in an online manner. The challenge is to solve a very large size problem instance in a very limited time. In order to do so, we formulate the problem using Constraint Programming (CP) as described in [10], and use Large Neighborhood Search (LNS) [15] to solve it. The basic idea of CP-based LNS is to repeatedly consider a subproblem, which defines a candidate neighborhood, and re-optimize it using CP. In the machine reassignment problem context, we select a subset of processes and reassign machines to them. In this paper we describe our CP-based LNS approach in detail.

There are several parameters to choose when implementing LNS, e.g., size of the neighborhood, when to change the neighborhood size, threshold in terms of time/failures for solving a subproblem etc. The values of these parameters can have a significant impact on the efficiency of LNS. We expose the parameters of our CP-based LNS approach, and study the impact of these parameters on LNS.

It is well known that manually tuning a parameterized solver can be very tedious and often consume a significant amount of time. Moreover, manual tuning rarely achieves maximal potential in terms of performance gains. Therefore, we study the application of a Gender-based Genetic Algorithm (GGA) for configuring the parameters automatically [2]. Experimental results show that the performance of the LNS solver tuned with GGA improves significantly, as com-

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

pared with the manually tuned LNS solver.[3] Furthermore, it is important to note that while tuning the parameters of GGA requires significant computational resources, it is still far faster than manual tuning. Additionally, GGA is an automated process that can be run in the background, thus releasing developers to focus their efforts on developing new algorithms rather than manually experimenting with parameter settings. Finally, the initial computational expenditure is further mitigated by the fact that the machine reassignment problem will be solved repeatedly in the future, so the costs tuning are amortized over time as the system is used in practice.

In the real world setting one can anticipate that the instances of the machine reassignment problem may differ from time to time. Thus, it is possible that one setting of parameters might not result in the best possible performance of the LNS solver across all possible scenarios. We, therefore, propose a methodology whereby in the offline phase a system continuously maintains a clustering of the instances and the LNS solver is tuned for each cluster of instances. In the online phase, when a new instance arrives the values of the parameters of the closest cluster are used for solving the instance. For this we study the application of Instance-Specific Algorithm Configuration (ISAC) [9]. Experimental results confirm that this further improves the performance of the LNS solver when compared with the solver tuned for all the instances with GGA. Overall the experimental results suggest that the proposed CP-based LNS approach with the aid of learning high quality parameter settings can find a good quality solution for a very large size instance in a very limited time.

The current computer industry trend is toward creating processor chips that contain multiple computation cores. If tuning the parameters of an LNS solver manually for single-core machine is tedious, then tuning for multiple parameterizations that would work harmoniously on multi-core machine is even more complex. We present an approach that can exploit multiple cores and can provide an order-of-magnitude improvement over manually configured parameters.

The paper is organized as follows. The machine reassignment problem is briefly described in Section 2 followed by the LNS used for solving this problem in Section 3. Section 4 describes how the parameters of LNS are tuned, and Section 5 presents experimental results followed by conclusions in Section 6.

2 Machine Reassignment Problem

In this section, we briefly describe the machine reassignment problem of ROADEF-EURO Challenge 2012 in collaboration with Google.[4] 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. Let o_p be the original machine on which process p is currently running. The objective is to find a solution that minimizes the cost of the reassignment.

[3] The manually tuned solver was runner up in the 2012 ROADEF-EURO Challenge and the difference between the first and the second was marginal.

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

In the following we describe the constraints, various types of costs resulting from the assignment, and the objective function.

2.1 Constraints

Capacity Constraints. 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 that 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, and a set of services partition the set of processes. The constraint is that the processes of a service should be assigned to different machines.
Spread Constraints. A location is a set of machines, and a set of locations partition the set of machines. These constraints ensure that processes of a service should be assigned to the machines such that their corresponding locations are spread over at least a given number of locations.
Dependency Constraints. A neighborhood is a set of machines and a set of neighborhoods also partition the machines. The constraint states that if service s depends on service s', then the set of the neighborhoods of the machines assigned to the processes of service s must be a subset of the set of the neighborhoods of the machines assigned to the processes of service s'.
Transient Usage Constraints. When a process is moved from one machine to another machine, some resources, e.g., hard disk space, are required in both source and target machines. These resources are called transient resources. The transient usage of a machine m for a transient-resource r is the sum of the amount of resource required by processes whose original or current machine is m. The transient usage of a machine for a resource should not exceed its capacity.

2.2 Costs

The objective is to minimize the weighted sum of load, balance, and move costs.
Load Cost. 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. To balance the availability of resources, a balance b is defined by a triple which consists of a pair of resources r_b^1 and r_b^2, and a multiplier t_b. For a given triple b, the balance cost is $\sum_{m \in M} \max(0, t_b \cdot A(m, r_b^1) - A(m, r_b^2))$ with $A(m, r) = c_{mr} - u_{mr}$.
Move Cost. A process move cost is defined as the cost of moving a process from one machine to another machine. The service move cost is defined as the maximum number of processes moved among services.

2.3 Instances

Table 1 shows the features of the machine reassignment problem and their limits on the instances of the problem that we are interested in solving. As this is

Table 1. Features of the problems instances.

Feature	Machines	Processes	Resources	Services	Locations	Neighbourhoods	Dependencies
Limit	5000	50000	20	5000	1000	1000	5000

an online problem, the challenge is to solve very large sized instances in a very limited time. Although the time limit for the competition was 300 seconds, we restrict the runtime to 100 seconds as we are solving more than 1000 instances with numerous parameter settings of the LNS solver.

3 Large Neighborhood Search

We formulated the machine reassignment problem using Constraint Programming (CP), which is described in [10]. We used Large Neighborhood Search for solving the instances of the problem. In this paper we omit the details of the CP model and we focus on the details of our LNS approach for solving this problem. In particular we focus on the parameters of the LNS solver that are carefully tuned for solving the problem instances efficiently.

LNS combines the power of systematic search with the scaling of local search. The overall solution method for CP-based LNS is shown in Figure 1. We maintain a current assignment, which is initialized to the initial solution given as input. At every iteration step, we select a subset of the processes to be reassigned, and accordingly update the domains of the variables of the CP model. We solve the resulting CP problem with a threshold on the number of failures, and keep the best solution found as our new current assignment.

3.1 Selecting a Subproblem

In this section we describe how a subproblem is selected. Our observation is that selecting a set of processes for reassignment from only some machines is better than selecting them from many machines. The reason is that if we select only a few processes from many machines, then we might not free enough resources

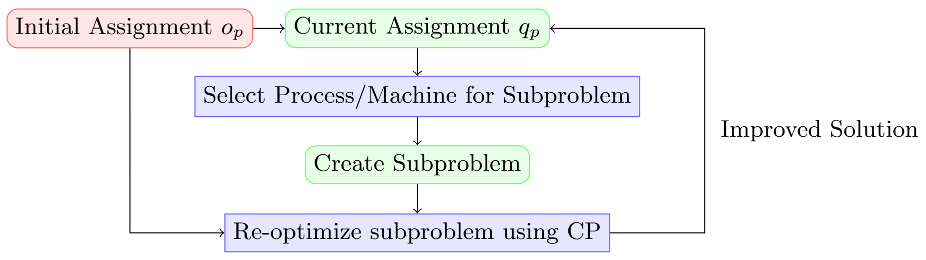


Fig. 1. Principles of the CP-based LNS approach

from the machines for moving the processes with large resource requirements from their original machines. Therefore, our subproblem selection is a two step process. First we select a set of machines and then, from each selected machine, we select a set of processes to be reassigned.

The number of machines that are selected in a given iteration is denoted by k_m. The maximum number of processes that are selected for reassignment from each selected machine is denoted by k_p. Both k_m and k_p are non-zero positive integers. The values of k_m and k_p can change during iterations of LNS. They are decided based on the several parameters of the LNS solver.

The number of processes that can be selected from one machine is bounded by an integer parameter, which is denoted by u_p. Therefore, $k_p \leq u_p$. The total number of processes that can be selected for reassignment is bounded by an integer parameter, which is denoted by t_p. Therefore, $k_m \cdot k_p \leq t_p$.

If all the processes of a machine are selected then many of them might be reassigned to their original machines again. Therefore, we enforce that the number of processes selected from a given machine should be less than or equal to some factor of the average number of processes on a machine. More precisely, $k_p \leq r_p \cdot (|P|/|M|)$. Here r_p is a continuos parameter.

Initially k_m is set to 1. As search progresses, it is incremented when a certain number of iterations in LNS are performed consecutively without any improvement in the quality of the solution [15]. The maximum value that can be set to k_m is denoted by t_m. We re-initialize k_m to 1 when it exceeds t_m. Depending on the value of k_m the value of k_p can change.

Notice that fewer processes on a machine implies fewer combinations of the processes that can be selected from the machine for reassignment and hence fewer possible subproblems that can be created from the selected machines. Therefore, the bound on the number of consecutive non-improving iterations is obtained by multiplying the average number of processes on a machine (i.e., $|P|/|M|$) by a continuous parameter, which is denoted by r_m. The value of r_m is bounded within 0.1 and 10. Notice that u_p, t_p, r_p, t_m, and r_m are constant parameters of the LNS algorithm and different values of these parameters can have a significant impact on the efficiency of CP-based LNS approach.

3.2 Creating the Subproblem

When solving a problem using LNS, the conventional way of creating a subproblem is to reinitialize all the domains of all the variables, reassign the machines to the processes that are not chosen for reassignment, and perform constraint propagation. This way of creating a subproblem can be a bottleneck for LNS when we are interested in solving a very large sized problem in a very limited time. Furthermore, if the size of the subproblem is considerably smaller than the size of the full problem then the number of iterations that one would like to perform will increase in which case the time spent in creating the subproblems will also increase further.

For example, let us assume that the total number of processes is 50000, the number of machines is 1000, and the maximum number of processes selected for reassignment for each iteration is 100. If we want to consider each process at least once for reassignment then we need at least 500 iterations. Assuming that the time-limit is 100 seconds, an average of 0.2 seconds will be spent on each iteration. Each iteration would involve selecting 100 processes for reassignment, reseting more than 50000 variables to their original domains, freezing 49900 variables, performing constraint propagation, and finally re-optimizing the subproblem. One can envisage that in this kind of scenario the time spent on creating subproblems can be a significant part of solving the problem.

This drawback is mainly because a CP solver is typically designed for systematic search. At each node of the search tree, a CP solver uses constraints to remove inconsistent values from the domains, and it uses trailing or copying with recomputation for restoring the values. Both trailing and copying with recomputation techniques are efficient for restoring the domains when an ordering is assumed on the way decisions are undone. However, in LNS one can move from one partial assignment to another by undoing any subset of decisions in an arbitrary order. Therefore, if an existing CP solver is used for LNS then un-assigning a set of processes would result in backtracking to the node in the search tree where all the decisions made before that node are still in place, and in the worst-case this could be the root node.

We propose a novel approach for creating the subproblem. The general idea is to use constraints to determine whether a removed value can be added to the current domain when a set of assignments are undone. Instead of using trailing or copying, we maintain two sets of values for each variable: (1) the set of values that are in the current domain of the variable, and (2) the set of values that are removed from the original domain. Additionally, we maintain two sets of variables: (1) a set of variables whose domains can be replenished, and (2) and a set of variables whose domains cannot be replenished. In each iteration, the former is initialized by the variables whose assignments are undone and the latter is initialized by the variables whose assignments are frozen. A variable whose domain cannot be replenished is also called a reduced variable.

The domain of a variable is replenished by checking the consistency of each removed value with respect to the constraints that involve reduced variables. Whenever a domain is replenished the variable is tagged as a reduced variable, its neighbors that are not reduced-variables are considered for replenishing their domains, and constraint propagation is performed on the problem restricted to the set of reduced variables. A fixed point is reached when no variable is left for replenishment. This approach is called replenishing the domains via incremental recomputation. Notice that existing CP-solvers do not provide support for replenishing domains via incremental re-computation. The main advantage of this approach is that it is not dependent on the order of the assignments and therefore it can be used efficiently to create subproblems during LNS.

3.3 Reoptimizing the Subproblem

We use systematic branch and bound search with a threshold on the number of failures for solving a given subproblem. The value of the threshold is obtained by multiplying the number of processes that are selected for reassignment with the value of a continuous parameter, which is denoted by t_f. The value of t_f ranges between 0.1 and 10. At each node of the search tree constraint propagation is performed to reduce the search space. The variable ordering heuristic used for selecting a process is based on the sum of the increment in the objective cost when assigning a best machine to the process and the total weighted requirement of the process which is the sum of the weighted requirements of all resources divided by the number of machines available for the process. The value ordering heuristic for selecting a machine for a given process is based on the minimum increment in the objective cost while ties are broken randomly.

4 Tuning Parameters of LNS

While it is possible to reason about certain parameters and their effect on the overall performance individually, there are numerous possible configurations that these parameters can take. The fact that these effects might not be smoothly continuous or that there may be subtle non-linear interactions between parameters complicates the problem further. Augment this with the time incurred at trying certain parameterizations on a collection of instances, and it becomes clear why one cannot be expected to tune the parameters of the solver manually.

Table 2. Parameters of LNS for Machine Reassignment Problem

Notation	Type	Range	Description
u_p	Integer	$[1, 50]$	Upper bound on the number of processes that can be selected from one machine for reassignment
t_p	Integer	$[1, 100]$	Upper bound on the total number of processes that can be selected for reassignment
r_p	Continuous	$[0.1, 1]$	Ratio between the average number of processes on a machine
t_m	Integer	$[2, 25]$	Upper bound on the number of machines selected for subproblem selection
r_m	Continuous	$[0.1, 10]$	Ratio between the upper bound on the consecutive non-improving iterations and the average number of processes on a machine
t_f	Continuous	$[0.1, 10]$	Ratio between the threshold on the number of failures and the total number of processes selected for reassignment

In the case of our LNS solver, Table 2 lists and explains the parameters that can be controlled. Although there are only six parameters, half of them are continuous and have large domains. Therefore, it is impractical to try all possible configurations. Furthermore, the parameters are not independent of each other. To test this, we gathered a small set of 200 problem instances and evaluated

400 randomly selected parameter settings on this test set. Then, using the average performance on the data set as our evaluation metric and the parameter settings as attributes, we ran feature selection algorithms from Weka [5]. All the attributes were found as important. Adding polynomial combinations of the parameter settings, further revealed that some pairs of parameters were more important than others when predicting expected performance.

Because of the difficulty of fully extracting the interdependencies of the parameters and covering the large possible search space, a number of automated algorithm configuration and parameter tuning approaches have been proposed over the last decade. These approaches range from gradient-free numerical optimization [3], gradient-based optimization [4], iterative improvement techniques [1], and iterated local search techniques like FocusedILS [7].

One of the more successful of these approaches is the Gender-based Genetic Algorithm (GGA) [2], a highly parallelizable tool that is able to handle continuous, discrete, and categorical parameters. Being a genetic algorithm, GGA starts with a large, random, population of possible parameter configurations. This population is then randomly split into two even groups: competitive and noncompetitive. The members of the competitive set are further randomly broken up into tournaments where the parameterizations in each tournament are raced on a subset of training instances. The best performing parameter settings of each tournament get the chance to crossover with the members of the noncompetitive population and continue to subsequent generations. In the early generations, each tournament has only a small subset of the instances, but the set grows at each generation as the bad parameter settings get weeded out of consideration. In the final generation of GGA, when all training instances are used, the best parameter setting has been shown to work very well on these and similar instances.

General tuning of a solver's parameters with tools like GGA, however, ignores the common finding that there is often no single solver that performs optimally on every instance. Instead, different parameter settings tend to do well on different instances. This is the underlying reason why algorithm portfolios have been so successful in SAT [8, 20], CP [12], QBF [14], and many other domains. These portfolio algorithms try to identify the structure of an instance beforehand and predict the solver that will have the best performance on that instance.

ISAC [9] is an example of a very successful non-model based portfolio approach. Unlike similar approaches, such as Hydra [19] and ArgoSmart [11], ISAC does not use regression-based analysis. Instead, it computes a representative feature vector in order to identify clusters of similar instances. The data is therefore clustered into non-overlapping groups and a single solver is selected for each group based on some performance characteristic. Given a new instance, its features are computed and it is assigned to the nearest cluster. The instance is then evaluated with the solver assigned to that cluster.

ISAC works as follows (see Algorithm 1). In the learning phase, ISAC is provided with a parameterized solver A, a list of training instances T, their corresponding feature vectors F, and the minimum cluster size κ. First, the gathered

Algorithm 1 Instance-Specific Algorithm Configuration

```
1: ISAC-Learn($A, T, F, \kappa$)
2: $(\bar{F}, s, t) \leftarrow$ Normalize($F$)
3: $(k, C, S) \leftarrow$ Cluster $(T, \bar{F}, \kappa)$
4: for all $i = 1, \ldots, k$ do
5:     $P_i \leftarrow GGA(A, S_i)$
6: return $(k, P, C, s, t)$
```

```
1: ISAC-Run($A, x, k, P, C, d, s, t$)
2: $f \leftarrow$ Features($x$)
3: $\bar{f}_i \leftarrow 2(f_i/s_i) - t_i \ \forall\, i$
4: $i \leftarrow \min_i(||\bar{f} - C_i||)$
5: return $A(x, P_i)$
```

features are normalized so that every feature ranges from $[-1, 1]$, and the scaling and translation values for each feature (s, t) is memorized. This normalization helps keep all the features at the same order of magnitude, and thereby keeps the larger values from being given more weight than the lower values. Then, the instances are clustered based on the normalized feature vectors. Clustering is advantageous for several reasons. First, training parameters on a collection of instances generally provides more robust parameters than one could obtain when tuning on individual instances. That is, tuning on a collection of instances helps prevent over-tuning and allows parameters to generalize to similar instances. Secondly, the found parameters are "pre-stabilized," meaning they are shown to work well *together*.

To avoid specifying the desired number of clusters beforehand, the g-means [6] algorithm is used. Robust parameter sets are obtained by not allowing clusters to contain fewer than a manually chosen threshold, a value which depends on the size of the data set. In our case, we restrict clusters to have at least 50 instances. Beginning with the smallest cluster, the corresponding instances are redistributed to the nearest clusters, where proximity is measured by the Euclidean distance of each instance to the cluster's center. The final result of the clustering is a number of k clusters S_i, and a list of cluster centers C_i. Then, for each cluster of instances S_i, favorable parameters P_i are computed using the instance-oblivious tuning algorithm GGA.

When running algorithm A on an input instance x, ISAC first computes the features of the input and normalize them using the previously stored scaling and translation values for each feature. Then, the instance is assigned to the nearest cluster. Finally, ISAC runs A on x using the parameters for this cluster.

In practice however, the tuning step of the ISAC methodology can be very computationally expensive, requiring on the order of a week on an 8 core machine. Fortunately it is not necessary to retune the algorithms too frequently. In many cases, even when new instances become available, the clusters are not likely to shift. We therefore propose the methodology shown in Figure 2. Here, given a set of initial instances, we perform the ISAC methodology to find a set of clusters for which the algorithm is tuned. When we observe new instances, we evaluate them according to the ISAC approach as shown in Figure 3, and afterwards add the instance to the appropriate cluster. But we also try re-clustering the entire set of instances. In most cases, the two clusterings will be similar, so nothing needs to be changed. But as we gather more data, we might see that one

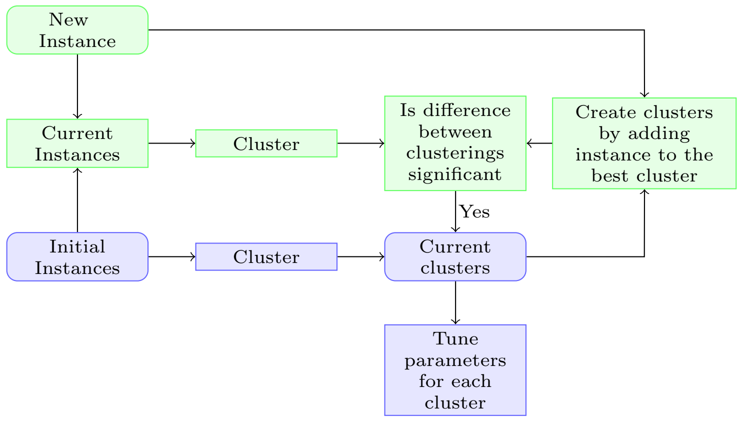

New Instance
Current Instances
Cluster
Is difference between clusterings significant
Create clusters by adding instance to the best cluster
Yes
Initial Instances
Cluster
Current clusters
Tune parameters for each cluster

Fig. 2. Offline Phase: Tuning LNS solver

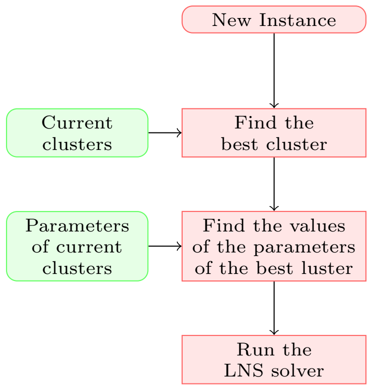

New Instance
Current clusters
Find the best cluster
Parameters of current clusters
Find the values of the parameters of the best luster
Run the LNS solver

Fig. 3. Online phase: Using tuned LNS solver for solving a given instance

of our clusters can be refined into two or more new clusters. When this occurs, we can then retune the LNS solver as needed.

5 Experimental Results

In this section we present results of solving the machine reassignment problem using CP-based LNS approach. We study three different ways of tuning the LNS solver. The first approach is the LNS solver, denoted by Default, which was runner up in the challenge. Here Default stands for a single set of parameters resulting from manual tuning of LNS solver on the 20 instances provided by 2012 ROADEF challenge organizers. The second approach is the LNS solver tuned using the GGA algorithm, which is denoted by GGA, and the final approach is the LNS solver tuned using ISAC, which is denoted by ISAC. The LNS solver for machine reassignment problem is implemented in C.[5]

In order to perform experiments in training and test phases we generated 1245 instances which were variations on the set B instances. Notice that the instances of set B are very large and they were used in the final phase of the ROADEF competition.[6] For each original instance we perturb the number of resources, the number of transient resources, the number of balances, weights of the load costs for resources, weights of balance costs, weights of process-move, machine-move cost and service-move costs. More precisely, for each original instance with $|R|$ number of resources, we randomly select k resources. The value of k is also chosen randomly such that $3 \leq k \leq |R|$. Out of k selected resources, a set of t resources are selected to be transient such that $0 \leq t \leq k/3$. The set of balances is also modified in a similar way. The original weight associated with each load-cost, balance-cost or any move-cost is randomly multiplied with a value selected randomly from the set $\{0.5, 1, 2\}$. Note that the uniform distribution is used to select the values randomly. All problems instances used in our experiments are available online.[7] The generated dataset was split to contain 745 training instances and 500 test instances. All the experiments were run on Linux 2.6.25 x64 on a Dual Quad Core Xeon CPU machine with overall 12 GB of RAM and processor speed of 2.66GHz.

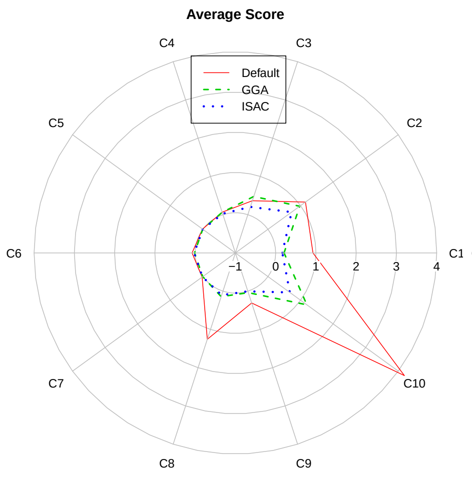
For evaluation of a solver's performance, we used the metric utilized for the ROADEF competition:

$$Score_S(I) = 100 * (Cost(S) - Cost(B))/Cost(R).$$

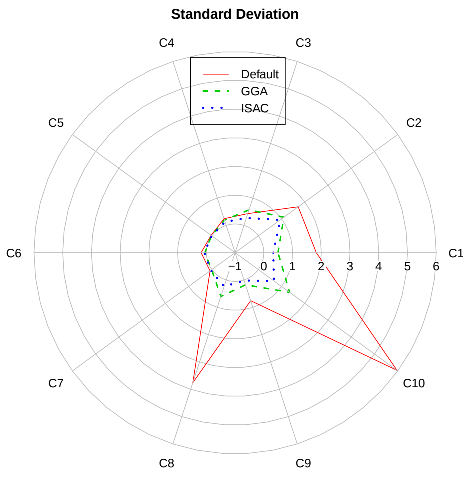
Here, I is the instance, B is the best observed solution using any approach, R is the original reference solution, and S is the solution using a particular solver. The benefit of this evaluation function is that it is not influenced by some instances having a higher cost than others, and instead focuses on a normalized value that ranks all of the competing approaches. We rely on using the best observed cost because for most of the instances it is not possible to find the optimal cost. On average, the best observed cost reduces the initial cost by 65.78%, and if we use the lower-bound then it reduces the initial cost by 66.98%. This demonstrates the effectiveness of our CP-based LNS approach for finding good quality solutions in 100 seconds. The lower-bound for an instance is obtained by aggregating the resource requirements over all processes and (safety) capacities over all machines and then computing the sum of the load and balance costs.

When tuning the LNS solver using GGA/ISAC, we used the competition's evaluation metric as the optimization criterion. However, to streamline the evaluations, we approximated the best performance using the performance achieved by running the LNS solver with default parameters for 1 hour. While this caused some of the scores to be negative during training, this approximation still correctly differentiated the best solver while avoiding placing more weight on in-

[5] `http://sourceforge.net/projects/machinereassign/`
[6] `http://challenge.roadef.org/2012/en/index.php`
[7] `http://4c.ucc.ie/~ymalitsky/`



(a) Mean scores of each cluster



(b) Standard deviations of each cluster

Fig. 4. Performance of the learned parameterizations from the Default, GGA, and ISAC medodologies.

stances with higher costs. In order to cluster the instances for ISAC, we used the features listed in Table 1. All these features are available in the problem definition so there is no overhead in their computation. When we clustered the instances using g-means with a minimal cluster size of 50, we found 10 clusters in our training data.

The performances of the learned parameterizations from the Default, GGA and ISAC methodologies is compared in Figure 4. Specifically, we plot the average performance of each method on the instances in each of the 10 discovered clusters (Figure 4(a)). What we observe is that even though the default parameters perform very close to as well as they can for clusters 4, 5, 6 and 7, for clusters 2, 8 and 10 the performance is very poor. Tuning the solver using GGA can improve the average performance dramatically. Furthermore, we see that if we focus on each cluster separately, we can further improve performance, highlighting that different parameters should be employed for different types of instances. Interestingly, we observe that ISAC also dramatically improves on the standard deviation of the scores (Figure 4(b)), suggesting that the tuned solvers are consistently better than the default and GGA tuned parameters.

Multi-Core Results. The current trend is to create computers with an ever increasing number of cores. It is unusual to find single core machines still in use, with 2, 4 or even 8 cores becoming commonplace. It is for this reason that we also experimented scenarios where more than one core is available for running the experiments. For Default and GGA, we ran the same parameters multiple times using different seeds, taking the best performance of 1, 2 and 4 trials. For ISAC however, we used the training data to pick which 1, 2, or 4 parameter set-

Table 3. Mean scores of the test data using Default, GGA, and ISAC approaches evaluated for 1, 2 and 4 cores. The standard deviations are presented in parentheses.

Approach	Number of Cores		
	1	2	4
Default	0.931 (2.759)	0.843 (2.596)	0.784 (2.541)
GGA	0.357 (0.808)	0.283 (0.663)	0.214 (0.529)
ISAC	**0.259 (0.623)**	**0.151 (0.363)**	**0.095 (0.237)**

tings associated with which clusters should be used for running the LNS solver in parallel. ISAC had the opportunity to choose from the 10 parameterizations found for each cluster plus the parameters of Default and GGA tuned solvers.

Table 3 shows that ISAC always dominates. While adding more cores is not particularly helpful for Default and GGA, ISAC can dramatically benefit from the additional computing power. And as can be seen from the reduction of the standard deviation, the ISAC tuned solvers are consistently better. Running t-tests on all the results, the benefits of GGA over default are always statistically significant (below 0.0001), as are the gains of ISAC over GGA. The detailed mean scores per cluster for various numbers of cores for ISAC are presented in Figure 5.

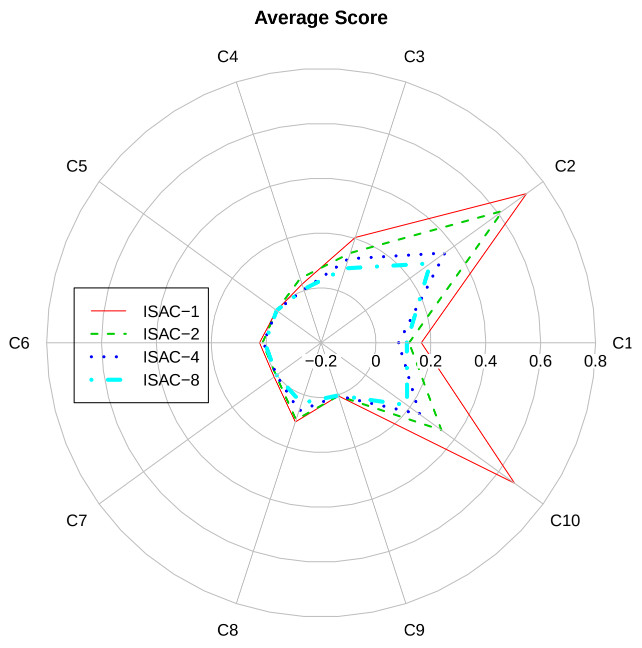


Fig. 5. Mean scores of each cluster ISAC approach for 1,2,4 and 8 cores.

Table 4. Average score on set B instances using Default, GGA, and ISAC trained parameterizations for 1, 2 and 4 cores. The standard deviations are in parentheses.

Approach	Number of Cores		
	1	2	4
Default	0.171 (0.268)	0.159 (0.261)	0.119 (0.211)
GGA	0.296 (0.416)	0.288 (0.417)	0.202 (0.285)
ISAC	**0.137 (0.224)**	**0.109 (0.184)**	**0.065 (0.120)**

Table 4 present results for 10 instances of set B which were used in the final phase of the competition. As the Default LNS is manually trained for set B instances, it is not surprising to see that the average score for set B is signficantly less than that obtained for 500 instances of test data. This demonstrates that the Default parameters have been over-tuned and because of that the performance of Default is poor on the test data. On the other hand the average score of GGA for test instances and for set B instances are very close. This demonstrates that the parameters of GGA are more stabilized, and therefore overall they work well for both test instances and set B instances. Table 3 shows that ISAC always dominates for set B instances. This confirms that for different types of instances different values of LNS parameters can help in solving problems more efficiently.

6 Conclusions

We have presented an effective constraint programming based Large Neighborhood Search (LNS) approach for the machine reassignment problem. Results show that our approach is scalable, and is suited for solving very large instances, and has good anytime behavior which is important when solutions must be reported subject to a time limit.

We have shown that by exposing parameters we are able to create an easily configurable solver. The benefits of such a development strategy are made evident through the use of automatic algorithm configuration. We show that an automated approach is able to set the parameters that out-perform a human expert. We further show that not all machine reassignment instances are the same, and that by employing the Instance-Specific Algorithm Configuration methodology we are able to improve the performance of our proposed approach. The tuning step is an initial cost that is quickly mitigated by the repeated usage of the learned parameters over a prolonged period of time.

Finally, we show that by taking advantage of the increasing number of cores available on machines, we can provide an order-of-magnitude improvement over using manually configured parameters.

References

1. Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Oper. Res.*, 54(1):99–114, January

2006.

2. Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In Ian P. Gent, editor, *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2009.
3. Charles Audet and Dominique Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. on Optimization*, 17(3):642–664, September 2006.
4. Steven P. Coy, Bruce L. Golden, George C. Runger, and Edward A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7:77–97, 2001.
5. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.
6. Greg Hamerly and Charles Elkan. Learning the k in k-means. In *In Neural Information Processing Systems*, page 2003. MIT Press, 2003.
7. Holger H. Hoos. *Autonomous Search.* 2012.
8. Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Proceedings of the 17th international conference on Principles and practice of constraint programming*, CP'11, pages 454–469, Berlin, Heidelberg, 2011. Springer-Verlag.
9. Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. Isac - instance-specific algorithm configuration. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.
10. Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Comparing solution methods for the machine reassignment problem. In Michela Milano, editor, *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 782–797. Springer, 2012.
11. Mladen Nikolić, Filip Marić, and Predrag Janičić. Instance-based selection of policies for sat solvers. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 326–340, Berlin, Heidelberg, 2009. Springer-Verlag.
12. Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of Artificial Intelligence and Cognitive Science (AICS 2008)*, 2008.
13. 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.
14. Luca Pulina and Armando Tacchella. A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th international conference on Principles and practice of constraint programming*, CP'07, pages 574–589, Berlin, Heidelberg, 2007. Springer-Verlag.
15. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1998.
16. Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of HotPower*, 2008.

17. 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.
18. 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.
19. Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.
20. Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *J. Artif. Int. Res.*, 32(1):565–606, June 2008.