Google Challenge: A Hyperheuristic for the Machine Reassignment Problem

Rodolfo Hoffmann*, María-Cristina Riff†, Elizabeth Montero‡ and Nicolás Rojas§
Department of Computer Science
Universidad Técnica Federico Santa María
Valparaíso, Chile
Email: *Rodolfo.Hoffman@inf.utfsm.cl, †Maria-Cristina.Riff@inf.utfsm.cl,
‡Elizabeth.Montero@inf.utfsm.cl, §NicolasRojas@acm.org

***Abstract*—In this work we present a hyperheuristic based method to solve Google Machine Reassignment problem (MRP). MRP was proposed at ROADEF/EURO challenge 2012 competition. It is a NP-complete problem. In the competition, this hard constrained optimization problem must be solved within 5 minutes. Our hyperheuristic approach uses a self-adaptive strategy according to the instance to solve, in order to quickly obtain quality solutions. The results show that self-adaptation is a good option for hyperheuristic approaches that require to be efficient in a restricted amount of time. Moreover, the results obtained using our hyperheuristic are competitive compared to those from the best algorithms of the competition.**

I. Introduction

In this work we present a hyperheuristic approach to the Machine Reassignment Problem (MRP) proposed in ROADEF/EURO challenge 2012. The MRP is defined as a set of machines and a set of processes requiring some resources from these machines. It is a constraint satisfaction optimization problem. At the beginning, the processes are assigned to the machines, the problem is to find a reassignment of machines to improve their current usage. Many constraints must be satisfied during the processes relocation step, for example machines availability, capacity, processes requirements, services relationship. Moreover, there are some costs associated to the reassignment of processes that must be taken into account to propose a new solution. In section II we detail the components, objectives and constraints of the Machine Reassignment Problem. In the challenge proposed by Google the reassignment must be done within 5 minutes. In the competition, instances grouped in two sets have been used to evaluate the proposed algorithms. In the first set, the number of processes ranges from 100 to $1,000$. The most difficult set is the second one, where the instances consider the number of processes ranging from $5,000$ to $50,000$.

Many metaheuristics have been proposed for this challenge. As far as we know, no hyperheuristic has been proposed in the final selection of the ROADEF/EURO challenge 2012. However, some simulated annealing based hyperheuristic approaches have been previously proposed [12], [3], [8]. We describe the best algorithms from the competition in section III. Hyperheuristics have been successfully applied to a wide range of combinatorial problems [2], [4], [11]. In this work, we propose a hyperheuristic approach that uses two low-level heuristics: The first one focuses its search on large neighborhoods and the second one focuses on small neighborhoods that show similar performance. We introduce our approach in section IV.

The contributions of this paper are:

- A hyperheuristic to solve the machine reassignment problem
- A dynamic selection strategy of the low-level heuristic that allows the self-adaptation of the hyperheuristic algorithm to the current instance.

We present an evaluation of each low-level heuristic, and we evaluate the benefits of using our self-adaptation strategy. We present these results and the statistical analysis in section V. We compare the performance of our hyperheuristic with the best algorithms from the competition. We present the conclusions and future work in section VI.

II. Machine Reassignment Problem

The Machine Reassignment Problem (MRP) was proposed at ROADEF/EURO challenge 2012[1]. The goal of MRP is to reassign a set of processes to a set of machines improving the machine usage. Each machine has a set of resources (RAM,CPU,..) and each process requires a certain capacity of these resources. Initially each process is assigned to a machine. In order to improve the quality of the initial assignment, processes can migrate to another machine. A process migration must satisfy hard constraints according to machine's capacities, their locations and processes requirements.

Next, we present the mathematical model of the MRP. It details the notation, hard constraints and the objectives defined in the problem.

† Supported by Fondecyt Project no. 1151456. Partially supported by the Centro Científico Tecnológico de Valparaíso (CCTVal) No. FB0821
‡ Supported by Postdoctoral Fondecyt Project no. 3130754

[1]http://challenge.roadef.org

A. Mathematical Model

1) Notation:

- $\mathcal{M}$ - set of machines
- $\mathcal{P}$ - set of processes
- $\mathcal{R}$ - set of resources
- $\mathcal{S}$ - set of services. A service corresponds to a set of processes $p \in \mathcal{P}$ which must run on different machines.
- $\mathcal{L}$ - set of locations. A location corresponds to a set of machines $m \in \mathcal{M}$. Locations are disjoint sets.
- $\mathcal{Z}$ - set of zones. A zone corresponds to a set of machines $m \in \mathcal{M}$. Zones are disjoint sets. In original problem formulation zones are called Neighborhoods.
- $\mathcal{TR}$ - set of transient resources. $\mathcal{R} \subseteq \mathcal{TR}$. A transient resource is a kind of resource consumed twice when a process is moved among machines; for example disk space is not available on machine during a copy from machine m to m', and m' should obviously have enough available disk space for the copy.

2) Decision variables:

- $M(p) = m$: Machine m performs process p. $M_0(p)$ corresponds to the initial assignment.

3) Constraints:

- **Capacity constraints**
  A process can run on a machine, if and only if, the machine has enough available capacity on every resource.
  $$\forall m \in \mathcal{M}, r \in \mathcal{R}, R(p,r) + U(m,r) \leq C(m,r)$$
  , where $C(m,r)$ is the capacity of resource $r \in \mathcal{R}$ for machine $m \in \mathcal{M}$; $R(p,r)$ is the requirement of resource $r \in \mathcal{R}$ for process $p \in \mathcal{P}$ and $U(m,r)$ is the usage of a machine m for a resource r.
- **Conflict constraints**
  Processes of each service must run on distinct machines.
  $$\forall s \in \mathcal{S}, (p_i, p_j) \in s^2, p_i \neq p_j \rightarrow M(p_i) \neq M(p_j)$$
- **Spread constraints**
  For each $s \in \mathcal{S}$ let $spreadMin(s) \in \mathrm{N}$ be the minimum number of distinct locations where at least one process of service s should run.
  $$\forall s \in \mathcal{S}, \sum_{l \in \mathcal{L}} min\Big(1, \Big|p \in s | M(p) \in l\Big|\Big) \geq spreadMin(s)$$
- **Dependency constraints**
  If service s^a depends on service s^b, then each process of s^a should run in the zone of a s^b process.

  $\forall p^a \in s^a, \exists p^b \in s^b$ and $z \in \mathcal{Z}$ such that $M(p^a) \in z$ and $M(p^b) \in z$
- **Transient usage constraints**
  Transient resources require capacity on both original assignment $M_0(p)$ and current assignment $M(p)$ for each resource needed.
  $$\forall m \in \mathcal{M}, r \in \mathcal{TR}, \quad \sum_{\substack{p \in \mathcal{P} \text{ such that} \\ M_0(p)=m \vee M(p)=m}} R(p,r) \leq C(m,r)$$

4) Objectives: The goal of the problem is to improve the usage of machines. This improvement considers the minimization of five different objectives:

- **Load cost**
  The load cost is defined per resource and corresponds to the used capacity above the safety capacity.
  $$loadCost(r) = \sum_{m \in \mathcal{M}} max(0, U(m,r) - SC(m,r))$$
  , where $SC(m,r)$ is the safety capacity of a resource $r \in \mathcal{R}$ on a machine $m \in \mathcal{M}$.
- **Balance cost**
  The balance cost tries to balance the use of available resources. In this case, the idea is to achieve a given target on the available ratio of two different resources.
  $$balanceCost(b) =$$
  $$\sum_{m \in \mathcal{M}} max(0, target \cdot A(m, r_1) - A(m, r_2))$$
  , where b corresponds to a triple $b = \langle r_1, r_2, target \rangle \in \mathcal{B}$ and $A(m,r) = C(m,r) - U(m,r)$.
- **Process move cost**
  Some process are painful to move. To model this soft constraint, a process move cost is defined.
  $$processMoveCost = \sum_{\substack{p \in \mathcal{P} \text{ such that} \\ M(p) \neq M_0(p)}} PMC(p)$$
  , where $PMC(p)$ corresponds to the cost of moving process p from its original machine.
- **Service move cost**
  To balance the moves among services, a service move cost is defined as the maximum number of moved processes over services.
  $$serviceMoveCost = max_{s \in \mathcal{S}}\Big(\Big|p \in s | M(p) \neq M_0(p)\Big|\Big)$$
  , where $PMC(p)$ corresponds to the cost of moving process p from its original machine.
- **Machine move cost**
  The machine move cost is the sum of all moves weighted by relevant machine move cost.
  $$machineMoveCost = \sum_{p \in \mathcal{P}} MMC(M_0(p), M(p))$$
  , where $MMC(m_{source}, m_{destination})$ corresponds to the cost of moving any process p from machine m_{source} to machine $m_{destination}$.

Finally, the total cost is:

$$\begin{aligned} totalCost &= \sum_{r\in\mathcal{R}} weight_{loadCost}(r)\cdot loadCost(r) \\ &+ \sum_{b\in\mathcal{B}} weight_{balanceCost}(b)\cdot balanceCost(r) \\ &+ weight_{processMoveCost}\cdot processMoveCost \\ &+ weight_{serviceMoveCost}\cdot serviceMoveCost \\ &+ weight_{machineMoveCost}\cdot machineMoveCost \end{aligned}$$

III. Related Work

Machine Reassignment Problem is a very complex NP-hard combinatorial problem [13]. The Google ROADEF/EURO challenge 2012: Machine Reassignment was attended by 82 teams divided into two categories: Junior (J) for young researchers and Senior (S) for qualified researchers.

The algorithms presented to the challenge were executed only once for each instance, with a maximum execution time of 5 minutes. The total score for each team corresponds to the sum of the scores for each instance. The winner of the Senior category was team $S41$ from University of Sarajevo in Bosnia using a Variable neighborhood search [5]. The winner of Junior category was team $J12$ from Poznan University of Technology in Poland using a hybrid MIP-based large neighborhood search [7].

Approach of team $S41$ [5] searches using different local search neighborhoods and uses a diversification strategy based on variations of the evaluation function. The local search starts from the given initial solution. Their main idea is to first reassign processes with a high level of requirements, because those processes will be more difficult to migrate later during the search. All the processes are sorted by absolute or relative requirements and reassignment is made considering only N processes. The local search procedure consists in exploring four neighborhoods: *Shift, Swap, Chain* and *BPR*. Shift migrates one process to another machine, Swap interchanges the assignments of two processes, Chain simultaneously shifts k processes, and BPR migrates process p to machine m and migrates some processes from m to another machines. The authors use the noisy method to improve the search diversification. It changes the objective function modifying the cost weight of a resource. This is repeated for few different resources, that are chosen by importance.

Team $S38$ [9] got the second place in Senior category. They use a constraint programming formulation specially suited for a large neighborhood search approach. The approach is a three step method that selects, creates and optimizes subproblems. k_m machines and k_p processes are selected. k_m cyclically goes from 1 to 10. The value of k_p is such that $k_m \cdot k_p \leq 40$. The problem is created emphasizing the efficiency of the algorithm to deal with large instances. It uses specially designed procedures to efficiently insert/remove processes in/from machines, as well as calculate the variations of the objectives. Re-optimization is performed using constraint programming considering three components: variable ordering heuristics, value ordering heuristics, and filtering rules.

Team $J12$ was the winner of the Junior category, but also the third place of the challenge. Team $J12$ approach consists of two components: a fast greedy hill climber and a large neighborhood search which uses mixed integer programming to solve subproblems [7]. Starting from the original assignment, the approach iteratively executes the hill climber to quickly improve the solution and then uses a MIP-based large neighborhood search to perform further improvements. The hill climber searches the neighborhood using a shift move which reassigns process p to another machine. The algorithm is deterministic and greedy, i.e. it accepts the first movement that lead to a feasible solution which cost is lower than the current one. If no better solution in the neighborhood is found, the algorithm stops. The second phase of the approach focuses on migrating several processes at once. As the size of the neighborhood grows exponentially with the number of processes and machines, the algorithm can be classified as a large neighborhood search. Exploring a neighborhood can be viewed as solving a subproblem of the original problem with a specialized procedure which can be a heuristic or an exact method. The subproblem is extracted from the original by selecting a subset of machines $\mathcal{M}_x \subset \mathcal{M}$. Given the current best solution, a mixed integer programming (MIP) solver is allowed to migrate any number of processes among machines in $\mathcal{M}_x$. The solver satisfies all the constraints given in the original problem, therefore it always produces a feasible solution. The acceptance criterion is also greedy. In this phase heuristics for *sup-problems selection*, *dynamic adaptation of sub-problem size* and *local search* are used to improve the performance of the approach.

IV. Our Approach

In this work we propose a hyperheuristic inspired on the simulated annealing metaheuristic [2], composed of two low-levels heuristics: $H0$ and $H1$. The main ideas of this proposal are:

- To allow the algorithm to work with different neighborhoods
- To allow the algorithm to select the most appropriate heuristic for each instance
- To allow on-line changes of the heuristic used.

We have designed heuristics $H0$ and $H1$ to deal with different scenarios. $H0$ uses moves to explore larger neighborhoods than $H1$. Thus, one iteration of $H0$ requires more time than one of $H1$. We use $H0$ when the current candidate solution has a significantly worse quality than its neighbors. When the quality of the neighbors of the current candidate solution is similar, then it is not worth to search in large neighborhoods. In the next sections we detail our hyperheuristic approach for the Machine Reassignment Problem $(HH-MRP)$.

A. $HH-MRP$

As simulated annealing does, $HH-MRP$ uses the temperature parameter to control the acceptation of new candidate solutions. Temperature $(T(t))$ is calculated at each step

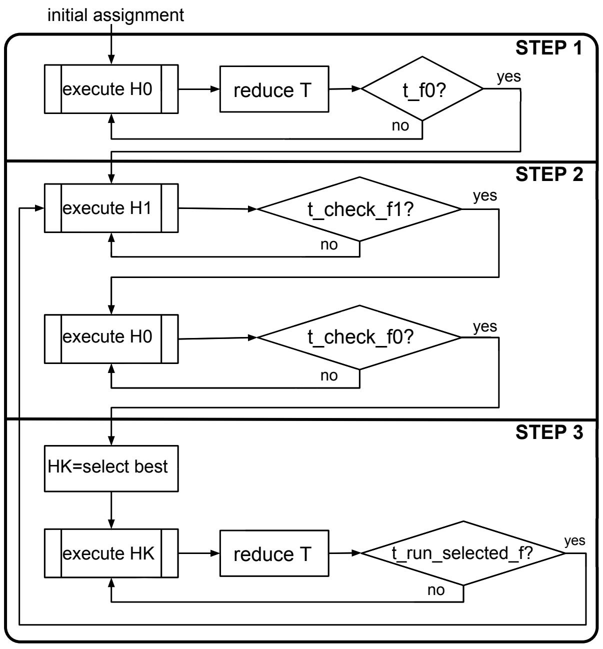

Fig. 1. HH-MRP global structure

according to equation 1.

$$T(t) = e^{temp\_control/(10 \cdot t + temp\_control/4.7)} - 10 \quad (1)$$

Figure 1 shows the global structure of HH-MRP.

As we mentioned before given an initial assignment (M_0) the goal is to find a better assignment by satisfying the involved constraints. The approach can be divided in three main steps, each one with the following characteristics:

- Step 1: In this step the hyperheuristic only uses heuristic H_0 beginning with the initial assignment (M_0). This step is done to explore the search space during a fixed time t_f0. The temperature value is modified. The candidate solution obtained (M) is the input of Step 2.
- Step 2: Given a candidate solution M, the hyper-heuristic independently uses $H0$ and $H1$ for a certain amount of time, t_check_f0 and t_check_f1 respectively. Each heuristic returns the best solution reached. The heuristic that produces the best solution is then selected. This step is a competitive one and its goal is to identify which of the two heuristics seems more appropriate to be used in the current state of the search. During this step, the temperature is not updated in order to evaluate both heuristics using the same conditions.
- Step 3: The hyperheuristic uses the best candidate solution obtained and the heuristic selected from Step 2 to continue the search process. When the stop condition for Step 3 is met ($t\_run\_selected\_f$ seconds), the candidate solution obtained here is used as entry for a new run starting in Step 2.

In this way our approach is able to dynamically self-adapt to the problem, executing periodically the more promising heuristic. Next, we explain in detail common components of the low-level heuristics $H0$ and $H1$.

B. Representation

The representation used for both heuristics is an array $x_1, x_2, \ldots, x_n$ which values belongs to $\{1, \ldots, m\}$, where n is the number of processes and m the number of machines. Thus, when $x_i = m$ means that process i is assigned to machine m.

C. Evaluation Function

Computing the objective function described in Section II-A for one candidate solution is very expensive. Because our hyperheuristic requires to evaluate the candidate solutions many times during its execution, we define the evaluation function $\Delta_{evaluation}$ as:

$$\begin{aligned}\Delta_{evaluation} &= \Delta_{loadCost}\\ &+ \Delta_{balanceCost}\\ &+ \Delta_{processMoveCost}\\ &+ \Delta_{serviceMoveCost}\\ &+ \Delta_{machineMoveCost}\end{aligned}$$

where each term corresponds to the variation of each associated cost. This evaluation function requires less computing time than the original objective function, without deteriorating its discrimination capability among different candidate solutions.

D. Moves

Our approach uses four moves: Insert, Swap, Double insert and Backtracking. Insert and Swap moves are usually applied to this kind of problems. We have detected that using only these moves is not sufficient for MRP to go through the search space, and thus more complex moves are required. We propose the Double insert move. Its goal is to transform an initial infeasible Insert move, that concerns one process and two machines, in a feasible one by migrating a process belonging to the target machine to a third one. We also propose the Backtracking move in order to locally improve the assignments of processes to machines. Only a subset of the processes and a subset of the machines are considered for backtracking,. The details of each move are:

- **Insert**
  Migrates a process from one machine to another one.
- **Swap**
  Selects two process p_1 and p_2 assigned to different machines $M(p_1) \neq M(p_2)$ and interchanges them.
- **Double insert**
  Selects a process p_1 from m_1 and tries to migrate it to machine $m_2 \in \mathcal{IM}_2$ ($\mathcal{IM}_2 \subseteq \mathcal{M}$). If it is not possible, the move tries to make space into m_2 by migrating a process p_2 from machine m_2 to machine $m_3 \in \mathcal{IM}_3$ ($\mathcal{IM}_3 \subseteq \mathcal{M}$). If it succeeds, then p_1 is migrated into m_2. Size of $\mathcal{IM}_2$ and $\mathcal{IM}_3$ are r_2max and r_3max respectively.

- **Backtracking (BT)**
  Randomly selects $n_{machines}$ and $n_{processes}$ and performs a backtracking process to this subproblem. The number of machines and processes is limited by parameter bt_num, where $n_{machines} \cdot n_{processes} \leq bt\_num$.

E. Low-level heuristics

In this section we describe in details the low-level heuristics that compose our hyperheuristic approach.

1) H0: The pseudocode of the low-level heuristic $H0$ is shown in algorithm 1. $H0$ uses three of the four moves: Insert, Swap and BT. According to a probability value (p_f0_m0) it chooses to either apply Insert or Swap move during a number of tries ($nh1\_try\_f0$ or $nh2\_try\_f0$ respectively). The resulting candidate solution is accepted when it is better than the current solution or when the probability that depends on the temperature allows to accept a deteriorated candidate solution. Finally, the BT move is applied to improve the best candidate solution found by using backtracking with a small set of randomly selected variables.

Algorithm 1: H0 heuristic

```
Input : $C \leftarrow$ current solution
1  $i \leftarrow 0$;
2  $B \leftarrow C$;
3  while $i \leq n\_c$ do
4      if $p\_f0\_m0$ then
5          $M \leftarrow evalMov(\textbf{Insert}, C, nh1\_try\_f0, nh1\_limit\_f0)$;
6      end
7      else
8          $M \leftarrow evalMov(\textbf{Swap}, C, nh2\_try\_f0, nh2\_limit\_f0)$;
9      end
10     $applyMov(M, T(t), \Delta_e(C), \Delta_e(B))$;
11     $i \leftarrow i + 1$;
12 end
13 if $p\_f0\_m3$ or no improvement in $C$ then
14     BT ($C$);
15 end
16 return $C$;
```

2) H1: The heuristic $H1$ uses three of the four moves: Insert, Swap and Double insert. The way that it uses Insert and Swap moves is similar to $H0$. The difference between the two heuristics is in the final process. $H1$ uses Double Insert in order to visit more candidate solutions from the current neighborhood. Algorithm 2 shows the pseudocode of $H1$.

Algorithm 2: H1 heuristic

```
Input : $C \leftarrow$ current solution
1  $B \leftarrow C$;
2  if $p\_f1\_m0$ then
3      $M \leftarrow evalMov(\textbf{Insert}, C, nh1\_try\_f1, nh1\_limit\_f1)$;
4  end
5  else
6      $M \leftarrow evalMov(\textbf{Swap}, C, nh2\_try\_f1, nh2\_limit\_f1)$;
7  end
8  $applyMov(M, T(t), \Delta_e(C), \Delta_e(B))$;
9  if $p\_f1\_m2$ then
10     Double insert ($C$);
11 end
12 return $C$;
```

V. Experimental Results

A. Experimental Setup

1) The Dataset: The organizers of the ROADEF/EURO challenge 2012 provided two set instances, A and B, 10 instances each. Set A contains small instances used for the qualification phase, while set B contains larger instances released to help teams with preparing algorithms for the final round. Table I summarizes the features of the instances in set B.

2) Experimental Environment: In the following experiments we allow each algorithm to try five times to solve each instance. In the ROADEF/EURO 2012 challenge, each algorithm was executed just one time, so we are giving them more opportunities to solve the problem. As ROADEF/EURO challenge established, each algorithm was run 5 minutes to solve each instance.
Our algorithm was implemented in C++ and executed using a Linux virtual machine on a QuadCore AMD FX-4300 with 4GB of RAM. The performance of our approach is always measured as the average of 5 independent executions. All the experiments where executed using the same seed as established in ROADEF/EURO challenge.

3) Parameter tuning: Tuning methods have been widely used during the last years [6], [1], [14]. Tuning methods are tools designed to determine the set of parameter values that optimizes the performance of metaheuristic methods. ParamILS [6] is an iterated local search based tuner that has shown a very competitive performance [10]. Starting from a default parameter calibration, ParamILS searches on its neighborhood looking for a parameter calibration of better quality. FocusedILS version of ParamILS is able to deal with the stochastic nature of metaheuristics using a comparison method that increases the number of executions for evaluating parameter calibrations when a more accurate comparison is required.

ParamILS tuner was used to tune the 21 parameters of the algorithm, 17 integer valued parameters and 4 real valued parameters. Tuning process was performed considering all the instances in set B and a total budget of 24 hours. For our experiments we used Frank Hutter's implementation of ParamILS available in his website[2]. Resulting parameter values are shown in table II.

[2]http://www.cs.ubc.ca/labs/beta/Projects/ParamILS

instance	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
processes	5,000	5,000	20,000	20,000	40,000	40,000	40,000	50,000	50,000	50,000
machines	100	100	100	500	100	200	4,000	100	1,000	5,000
resources	12	12	6	6	6	6	6	3	3	3
services	2,512	2,462	15,025	1,732	35,082	14,680	15,050	45,030	4,609	4,896
neighborhoods	5	5	5	5	5	5	5	5	5	5
dependencies	4,412	3,617	16,560	40,485	14,515	42,081	43,873	15,145	43,437	47,260
locations	10	10	10	50	10	50	50	10	100	100
balance triples	0	1	0	1	0	1	1	0	1	1

TABLE I. CHARACTERISTICS OF INSTANCES IN SET B

parameter	description	value
t_f0	time for $H0$ at STEP 1	40
$t\_run\_selected\_f$	time for HK at STEP 3	15
t_check_f0	time to check $H0$ at STEP 2	3
t_check_f1	time to check $H1$ at STEP 2	4
$temp\_control$	control reduction of temperature	850
n_c	number of movements in $H0$	3
p_f0_m0	probability of using **Insert** in $H0$	0.7
p_f0_m3	probability of using **BT** in $H0$	0.15
p_f1_m0	probability of using **Insert** in $H1$	0.7
p_f1_m2	probability of using **Double insert** in $H1$	0.1
r_2max	# machines in 1st step of **Double insert**	100
r_3max	# machines in 2nd step of **Double insert**	10
bt_num	Size limit of subproblem for **BT**	215
$nh1\_try\_f0$	Neighborhood limit for **Insert** in $H0$	20600
$nh1\_limit\_f0$	Steps w/o improvements for **Insert** in $H0$	2060
$nh2\_try\_f0$	Neighborhood limit for **Swap** in $H0$	20600
$nh2\_limit\_f0$	Steps w/o improvements for **Swap** in $H0$	4060
$nh1\_try\_f1$	Neighborhood limit for **Insert** in $H1$	600
$nh1\_limit\_f1$	Steps w/o improvements for **Insert** in $H1$	30
$nh2\_try\_f1$	Neighborhood limit for **Swap** in $H1$	600
$nh2\_limit\_f1$	Steps w/o improvements for **Swap** in $H1$	90

TABLE II. PARAMETER VALUES

B. Results

In this section we present the experimental results of our hyperheuristic approach. The first experiment is a comparison among the low-level heuristics and our hyperheuristic. The second experiment compares our approach with the best results of the ROADEF/EURO challenge 2012. The performance shown corresponds to the relative error to the best result found for each instance.

1) Hyperheuristic approach: In table III we compare the performance obtained by each low-level hyperheuristic ($H0$ and $H1$), the sequential combination of both ($H0\&H1$) and the hyperheuristic proposed $HH-MRP$. In table III we can observe that both $H0$ and $H1$ show a comparable performance for instances B2, B3, B4, B6 and B9, but in the remaining instances the differences are important. We can also observe that using both low-level heuristics (third column) produces an average performance, which is reasonable because low-level heuristics are good for certain stages of the search and when they work together they can collaborate. Despite this, the best performance in these experiments was obtained using $HH-MRP$. In 9 of 10 instances, our hyperheuristic obtained the best performance. Hence, we can conclude that the hyperheuristic proposed is able to self-adapt the low-level heuristics using its dynamic selection strategy.

Statistical comparisons of these results using Wilcoxon tests are shown in table IV. For these comparisons we considered for each algorithm the performance of 5 executions for each instance. Here we can observe that the performance of the hyperheuristic proposed is statistically different than low-level heuristic's performance and the sequential combination of these two low-level heuristics.

comparison			p-value
$HH-MRP$	vs	$H0$	0.00
$HH-MRP$	vs	$H1$	0.00
$HH-MRP$	vs	$H0\&H1$	0.00

TABLE IV. WILCOXON COMPARISON $HH-MRP$ VERSUS LOW-LEVEL HEURISTICS

2) Comparison with ROADEF/EURO results: With these experiments we aim to compare our approach with the results obtained in the ROADEF/EURO challenge 2012. First, we compare our results with the average results obtained by the best 16 Senior teams competing and next, we compare our performance against the results of the teams of first three places in Senior category.

Table V shows the average, best and worst results for each instance obtained by the best 16 Senior teams published in ROADEF/EURO challenge 2012 results, last row shows our average performance in each instance. Here, we observe that for each instance, there is a big difference, up to 13%, between best and worst results and a difference of up to 3% between the best and the average result in the challenge. Our proposal is always better than the average and shows a difference of at most 1% respect to the best result in each instance.

In table VI we compare the performance obtained by the three first places in Senior category in the ROADEF/EURO challenge 2012 and the hyperheuristic proposed $HH-MRP$. In order to perform a proper comparison, available source codes of these teams were downloaded and executed all in the same machine. The performance of these algorithms is also measured as the average of 5 executions. Here, it is possible observe that the hyperheuristic proposed shows on average a 0.24% worse performance than teams $S41$ and $S38$ and a 0.02% better performance than team $S14$. From these results it is important to notice that there is no a unique winner in all the instances tested although the best results were most times found by teams $S41$ and $S38$.

Table VII shows the results of the Wilcoxon tests comparing our approach with the results of the three best teams presented in senior category in ROADEF/EURO challenge 2012. For these tests we considered for each algorithm the performance of 5 runs for each of 10 instances. Here we can observe that the performance of teams $S41$ and $S38$ are

instance	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
$H0$	55.37	**79.83**	97.30	**49.21**	89.39	**25.29**	57.07	88.28	**31.63**	51.56
$H1$	50.78	79.24	97.46	**49.21**	92.46	**25.29**	59.59	91.02	**31.63**	55.70
$H0\&H1$	54.81	79.64	97.45	**49.21**	91.07	**25.29**	58.81	90.39	**31.63**	53.71
$HH-MRP$	**55.40**	79.77	**97.49**	**49.21**	**92.56**	**25.29**	**60.48**	**91.34**	**31.63**	**56.48**

TABLE III. COMPARISON $HH-MRP$ VERSUS LOW-LEVEL HEURISTICS

instance	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
$bestROADEF$	**56.32**	**80.40**	**97.53**	49.20	**92.57**	**25.29**	**60.91**	**91.37**	**31.63**	**57.25**
$worstROADEF$	43.31	74.35	93.27	48.58	90.61	25.12	53.64	88.78	30.28	42.42
$averageROADEF$	52.50	79.59	96.93	49.13	92.28	25.27	59.84	91.03	31.46	55.45
$HH-MRP$	55.40	79.77	97.49	**49.21**	92.56	**25.29**	60.48	91.34	**31.63**	56.48

TABLE V. COMPARISON $HH-MRP$ VERSUS 16 PARTICIPANTS IN SENIOR CATEGORY IN ROADEF/EURO

instance	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Average
$S41$	55.89	**80.40**	**97.52**	**49.21**	**92.57**	**25.29**	**60.91**	**91.37**	**31.63**	**57.25**	**64.20**
$S38$	**55.93**	80.13	**97.52**	**49.21**	**92.57**	**25.29**	60.87	**91.37**	**31.63**	57.22	64.17
$S14$	54.67	80.29	97.04	**49.21**	92.02	**25.29**	60.76	91.28	**31.63**	**57.25**	63.94
$HH-MRP$	55.40	79.77	97.49	**49.21**	92.56	**25.29**	60.48	91.34	**31.63**	56.48	63.96

TABLE VI. COMPARISON $HH-MRP$ VERSUS THREE FIRST PLACES IN ROADEF/EURO

comparison			p-value
$HH-MRP$	vs	$S41$	0.00
$HH-MRP$	vs	$S38$	0.00
$HH-MRP$	vs	$S14$	**0.86**

TABLE VII. WILCOXON COMPARISON $HH-MRP$ VERSUS THREE FIRST PLACES IN ROADEF/EURO

statistically better than the performance of our approach, but $HH-MRP$ performs statistically equal to the third place in senior category in challenge, team $S14$.

VI. CONCLUSIONS AND FUTURE WORK

In this work we have presented a hyperheuristic approach for the Machine Reassignment Problem proposed in ROADEF/EURO challenge 2012. Our approach was designed in order to perform properly for different instances and to find good results within five minutes. Our proposal uses two low-level heuristics oriented to deal with different scenarios: $H0$ uses moves to explore large neighborhoods and $H1$ to deal with neighborhoods where the quality of solutions is similar, thus $H0$ requires more time than $H1$. In this work, we focused on designing a hyperheuristic able to select the most appropriate heuristic for each instance and able to perform on-the-fly changes of the heuristic used.

Our tests were performed considering the same conditions than the ROADEF/EURO challenge 2012. From our results, we can conclude that our proposal was able to adapt its behavior to the step of the search and to the instance being solved, showing a statistically better performance than each low-level heuristic and a simple sequential combination of them both. Moreover, compared with the results of the challenge, our proposal was able to show a performance statistically comparable to the third best competitor in the senior category of the challenge.

As future work we plan to evaluate the inclusion of heuristics that have shown good results in some other approaches from state of the art. These heuristics, based on features of process and machines, could help to improve the effectiveness of the search in neighborhoods, and therefore, improve the quality of results the algorithm can find with an upper bound of 5 minutes.

REFERENCES

[1] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer Berlin Heidelberg, 2010.

[2] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[3] Edmund K. Burke, Graham Kendall, Mustafa Misir, and Ender Özcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1):73–90, 2012.

[4] Pablo Garrido and María-Cristina Riff. Dvrp: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16(6):795–834, 2010.

[5] Haris Gavranović, Mirsad Buljubašić, and Emir Demirović. Variable neighborhood search for google machine reassignment problem. *Electronic Notes in Discrete Mathematics*, 39:209 – 216, 2012. {EURO} Mini Conference.

[6] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second Conference on Artifical Intelligence*, pages 1152–1157, 2007.

[7] Wojciech Jaśkowski and Marcin Szubert andh Piotr Gawron. A hybrid mip-based large neighborhood search heuristic for solving the machine reassignment problem. *Annals of Operations Research*, 2015. (accepted).

[8] Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing*, 17(12):2279–2292, 2013.

[9] Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Comparing solution methods for the macine reassignment problem. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, pages 782–797, Berlin, Heidelberg, 2012. Springer-Verlag.

[10] Elizabeth Montero, María-Cristina Riff, and Bertrand Neveu. A beginner's guide to tuning methods. *Applied Soft Computing*, 17:39–51, 2014.

[11] Gabriela Ochoa, Rong Qu, and Edmund K. Burke. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems.

In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 341–348. ACM, 2009.

[12] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[13] Gabriel Marques Portal. *An algorithmic study of the machine reassignment problem*. PhD thesis, Universidade Federal do Rio Grande do Sul, 2012.

[14] María-Cristina Riff and Elizabeth Montero. A new algorithm for reducing metaheuristic design effort. In *IEEE Congress on Evolutionary Computation (CEC 2013)*, pages 3283–3290, 2013.