IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 9, 2020, accepted April 27, 2020, date of publication May 11, 2020, date of current version May 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993563

A Survey and a Classification of Recent Approaches to Solve the Google Machine Reassignment Problem

DARIO CANALES, NICOLÁS ROJAS-MORALES, (Member, IEEE), AND MARÍA-CRISTINA RIFF
Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso 2390123, Chile

Corresponding author: Nicolás Rojas-Morales (nicolas.rojasm@usm.cl)

This work was supported in part by the Universidad Técnica Federico Santa María (UTFSM) under Project PI_LI_19_16, and in part by the Fondo Nacional de Desarrollo Científico y Tecnológico (FONDECYT) Project under Grant 1200126.

ABSTRACT Optimizing the usage of resources is an important topic in the development of technologies and computational services. The Google Machine Reassignment Problem is an NP-hard problem that is related to this crucial situation, based on the assignation of a set of processes into a set of machines trying to reduce several costs. This problem was proposed for the 2012 ROADEF/EURO challenge and since its introduction, many approaches have been proposed in order to reach better quality solutions or improve the execution time of the existing techniques. In this work, we review a significant number of recently proposed approaches. Due to the number of published papers, it is difficult to ascertain the level of current research in this area. In order to provide a useful guide to new interested researchers, we include up-to-date best-known results for benchmark instances, an analysis of the design of each technique and details of the experimental setup. We also present a classification and a taxonomy of the reviewed approaches based on the design of these techniques, considering their main components and the structure of the search strategies.

INDEX TERMS Google machine reassignment problem, metaheuristics, heuristics.

I. INTRODUCTION

Recently, the emerging development of technologies that provide cloud services and data storage have generated great interest in optimizing the usage of resources, which has a direct impact on the service quality and the economy of this business. A key aspect to consider services such as Google Apps, Facebook and Microsoft is the maximization of the obtained utilities along with the minimization of the operative costs. There are many situations that may impact on the performance of a computational service and as a consequence, many strategies have been used to reduce the impact of failures and eventual shutdowns. Additionally, the overuse of the resources of a machine may incur in additional costs, related to its deterioration or to a greater electrical consumption.

Along the same line, the increasing concern about the environmental impact of human activities is an important motivation to reduce the electric energy consumption [1], [2]. Considering that only on the United States the 29% of the greenhouse gases in 2015 where due to the electricity generation [3], the interest of reducing the greenhouse gas emissions to compensate the climate change has increased.

These crucial situations are related to the Google Machine Reassignment Problem (GMRP), the main topic of this article, proposed by Google in 2012 in the context of the ROADEF/EURO challenge. The objective of this problem is to manage and re-assign a set of processes into a set of machines, considering a set of hard-to-satisfy constraints. The GMRP is a NP-hard problem and several approaches have been proposed after the challenge on 2012 [4]. Section II presents a description and the main details of the problem.

The objective of this article is to present an up-to-date revision of the recently proposed approaches for solving this interesting problem. Considering the number of proposed approaches for the GMRP, it can be difficult to obtain a broad view to understand the existing techniques, the trends in the design of approaches for the GMRP, the actual best known results, among other features. Our main motivation is to provide an useful guide to new researchers that are interested in working in the GMRP. For each technique, we considered

The associate editor coordinating the review of this manuscript and approving it for publication was Yan-Jun Liu.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

their main algorithmic features, the instances used to evaluate each approach and the performance of each algorithm. On the ROADEF/EURO competition, a set of experimental setup rules were defined. Considering the most used benchmark instances of the GMRP, we report the actual best known quality solution per instance in two scenarios: considering the same rules of the competition or using a different experimental setup. Moreover, we report which algorithms are able to reach each solution in both scenarios. We also present details and a comparative of the experimental setup of each reviewed technique with the hardware of the ROADEF/EURO competition. Section III presents the revision of techniques and Section IV presents an analysis and a discussion of the reviewed approaches.

We propose a classification and a taxonomy of the reviewed techniques considering design characteristics, algorithmic features, and some characteristics related to the proposed strategies for solving this complex problem.

The main contributions of this article are:

- A revision of the existing approaches proposed for solving the GMRP since the ROADEF/EURO challenge
- An updated list of the actual best known solutions per benchmark instance
- A classification and taxonomy of the reviewed techniques

It is important to mention that there are some previously published literature reviews that are focused in a different manner that our work (e.g, a revision and classification of the finalist teams of the ROADEF/EURO challenge is presented in [5]). Here, we presented an up-to-date revision, classification and taxonomy of algorithms proposed for the GMRP. Section V presents the classification and taxonomy. Finally, Section VI presents some conclusions of this revision and trends for future work.

II. MACHINE REASSIGNMENT PROBLEM

The Google Machine Reassignment Problem (GMRP) was formally presented for the ROADEF/EURO Challenge 2012 [6]. This problem consists of the allocation of a series of processes to a set of machines satisfying a set of constraints. A feasible initial assignment s_0 is provided in each problem instance, where all the processes are distributed among the machines. To determine the quality of a candidate solution, the objective function considers a weighted sum of a Load Cost, a Balance Cost, a Process Move Cost, a Service Move Cost, and a Machine Move Cost. The problem also considers a set of constraints that are related to the capacity of the machines, the usage of resources, dependency, among others. For example, each process has system requirements of various resources like CPU, RAM and Hard Disk. On the other hand, machines have a certain capacity limit for each resource. These resources are consumed by each assigned process. Starting from s_0, the objective is to obtain a better quality candidate solution that minimizes the different costs of the problem, as well as satisfying all constraints. This section presents a brief description of the GMRP and in order to complement this explanation, on the Annexes section, a mathematical model of the MRP is presented.

TABLE 1. Hard constraints summary.

Constraints	Details	Focus
Capacity	Resource capacities of each machine should not be exceeded	Resource requirements feasibility
Conflict	Processes of the same service should not be assigned to a same machine	Robustness to possible machine failures
Spread	Some processes, related to some services, should be assigned to a minimum number of different locations	Robustness to possible machine failures
Dependency	Dependent services should be executed on the same neighborhood	Robustness to possible machine failures
Transient Usage	Defines that some resources require a transient usage when processes are moved between machines	Retain resources during migration of processes

A. COMPONENTS

The Google Machine Reassignment Problem (GMRP) consider the following components:

- ***Processes:*** elements that need to be assigned considering their resource requirements.
- ***Services:*** can be considered as sets of processes. Services have dependence relationships, which are traduced into constraints of the problem.
- ***Machines:*** equipment to which processes will be assigned. Each machine has a set of available resources with a defined capacity. These resources will be consumed by some assigned processes.
- ***Location:*** is defined as a set of machines. A location defines how processes of a particular service should be distributed (among machines of different locations). Moreover, for each service, a minimum number of locations where their processes should be assigned is defined (*spreadmin*). All locations are disjoint sets.
- ***Neighborhood:*** is defined as a set of machines. When a dependence relationship between services exists, processes should be assigned on machines of the same neighborhood. All neighborhoods are disjoint sets.

The problem constraints are grouped into five main categories: Capacity, Conflict, Spread, Dependency and Transient Usage. Table 1 presents a summary of these constraints. The general objective of the GMRP is to aim for a better distribution of processes on the available machines. Following this idea, the objective function of the GMRP minimizes the weighted sum of five cost functions summarized in Table 2.

1) EXAMPLES

Let us suppose a small problem instance that contains a set of processes (p_1, p_2, p_5), a set of machines (m_A, m_B, m_C, m_D), a service $s_\&$ (that considers processes p_2 and p_5), a location $l_\#$ (that considers machines m_A and m_B) and a neighborhood

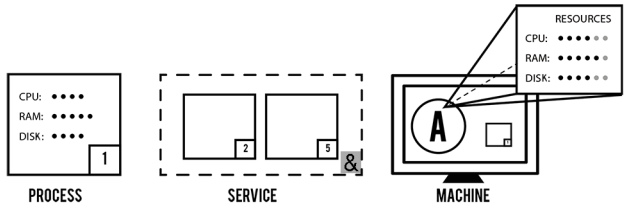

FIGURE 1. Components of GMRP.

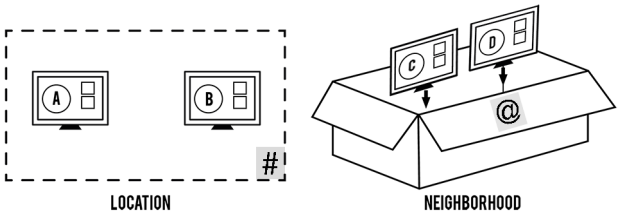

FIGURE 2. Components that describe sets of machines in the problem.

TABLE 2. Costs considered in GMRP.

Costs	Details
Load Cost	Considering some defined *safety capacity* boundaries, it represents the excess of used resources on each machine.
Balance Cost	Considering some defined *resource balance* rules, it represents how the resources are consumed on each machine.
Process Move Cost	Considering the provided initial assignment s_0, it represents the cost of migrate processes from their original machines
Service Move Cost	Considering all the moved processes, it represents the maximum number of moved processes over services
Machine Move Cost	Represents the cost of moving a process from a source machine to a destination machine

$N_@$ (that considers m_C and m_D). Figures 1 and 2 present these components and their relationships. Machine m_A has 4 available units of CPU, 5 units of RAM and 4 units of Disk. Here, process p_1 can be assigned to m_A because its requirements are fully satisfied (satisfying the capacity constraint).

Let us suppose that a transient usage $TR = \{Disk\}$ has been defined. This means that when processes are reassigned, consumed units of *Disk* are not available to be used by other processes until the next iteration (or reassignment step). For example, if p_1 is moved from machine m_A, the 4 consumed units of *Disk* will not be available until the next iteration. As a consequence, processes that consume more than 2 units of *Disk* (and more than 6 units of CPU and 6 units of RAM), will not be able to be assigned to m_A at this step.

Let us suppose we want to assign p_2 and p_5 to a machine. As these processes are part of the service $s_\&$, the conflict constraint will be violated if both processes are assigned to the same machine (e.g. machine m_A). This constraint requires that p_2 and p_5 should be assigned to different machines.

The spread constraint requires that some processes, related to a service, should be assigned to a minimum number of different locations (named *SpreadMin*). Let us suppose that a *SpreadMin* value is defined as 2. The spread constraint will be violated if we assign p_2 and p_5 to machines m_A or m_B, considering that these machines are part of $l_\#$. The constraint will force the assignation of both processes in 2 different locations.

Let us suppose that we have a new service $s_=$ and suppose that $s_=$ depends on $s_\&$. The dependency constraint forces that processes of both services should be assigned to machines of the same neighborhood. In this case, if p_2 and p_5 are assigned to machine m_C, processes of $s_=$ should be also assigned to machines m_C or m_D.

B. DETAILS OF THE COMPETITION

The Machine Reassignment Problem was proposed in ROADEF/ EURO 2012 [6].[1] On the competition, 48 teams participate of the qualification phase. Teams were divided in two categories: Junior (for young researchers) and Senior (for qualified researchers).

1) EXPERIMENTAL SETUP OF THE CHALLENGE

In order to evaluate each presented algorithm, an independent execution was performed sequentially with a time limit of 300 seconds. All experiments were executed on a computer with an Intel Core 2 Duo E8500 (3.16 GHz, 64 bits, 4 GB of RAM).

2) INSTANCES

In the competition, 30 instances were used to evaluate all the proposed approaches. These instances are available in the website of the challenge and are divided in three sets:[2]

- Set A: problem instances that consider a limit of 1000 processes.
- Set B: problem instances with a number processes between 5000 and 50000.
- Set X: problem instances with a number processes between 5000 and 50000.

Tables 3, 4 and 5 show the main characteristics of the instances of the data sets A, B and X respectively. Nowadays, these instances have been widely used as benchmark problems to evaluate and compare different algorithms in the literature.

3) SCORE AND PHASES

A specific metric to rank each algorithm was defined, using the following score:

$$Score(I) = \frac{Cost(S) - Cost(B)}{Cost(s_0)} \cdot 100 \qquad (1)$$

where *Cost* of the evaluation of a candidate solution, I is the problem instance in which the score will be obtained, s_0 is the initial solution, S the solution found by the algorithm and B is the best solution found in the challenge among all algorithms for that specific instance. The two phases of the ROADEF competition were:

[1]Competition webpage http://challenge.roadef.org/2012/en/
[2]Competition web-page http://challenge.roadef.org/2012/en/instances.php

TABLE 3. Data set A characteristics - considering $|\mathcal{R}|$ the number of resources, $|\mathcal{M}|$ the number of machines, $|\mathcal{S}|$ the number of services, $|\mathcal{P}|$ the number of processes, $|\mathcal{L}|$ the number of locations and $|\mathcal{N}|$ the number of neighborhoods.

Instance	$\|\mathcal{R}\|$	$\|\mathcal{M}\|$	$\|\mathcal{S}\|$	$\|\mathcal{P}\|$	$\|\mathcal{L}\|$	$\|\mathcal{N}\|$
A1_1	2	4	79	100	4	1
A1_2	4	100	980	1,000	4	2
A1_3	3	100	216	1,000	25	5
A1_4	3	50	142	1,000	50	50
A1_5	4	12	981	1,000	4	4
A2_1	3	100	1,000	1,000	1	1
A2_2	12	100	170	1,000	25	5
A2_3	12	100	129	1,000	25	5
A2_4	12	50	180	1,000	25	5
A2_5	12	50	153	1,000	25	5

TABLE 4. Data set B characteristics - considering $|\mathcal{R}|$ the number of resources, $|\mathcal{M}|$ the number of machines, $|\mathcal{S}|$ the number of services, $|\mathcal{P}|$ the number of processes, $|\mathcal{L}|$ the number of locations and $|\mathcal{N}|$ the number of neighborhoods.

Instance	$\|\mathcal{R}\|$	$\|\mathcal{M}\|$	$\|\mathcal{S}\|$	$\|\mathcal{P}\|$	$\|\mathcal{L}\|$	$\|\mathcal{N}\|$
B_1	12	100	2,512	5,000	10	5
B_2	12	100	2,462	5,000	10	5
B_3	6	100	15,025	20,000	10	5
B_4	6	500	1,732	20,000	50	5
B_5	6	100	35,082	40,000	10	5
B_6	6	200	14,680	40,000	50	5
B_7	6	4,000	15,050	40,000	50	5
B_8	3	100	45,030	50,000	10	5
B_9	3	1,000	4,609	50,000	100	5
B_10	3	5,000	4,896	50,000	100	5

TABLE 5. Data set X characteristics - considering $|\mathcal{R}|$ the number of resources, $|\mathcal{M}|$ the number of machines, $|\mathcal{S}|$ the number of services, $|\mathcal{P}|$ the number of processes, $|\mathcal{L}|$ the number of locations and $|\mathcal{N}|$ the number of neighborhoods.

Instance	$\|\mathcal{R}\|$	$\|\mathcal{M}\|$	$\|\mathcal{S}\|$	$\|\mathcal{P}\|$	$\|\mathcal{L}\|$	$\|\mathcal{N}\|$
X_1	12	100	2,529	5,000	10	5
X_2	12	100	2,484	5,000	10	5
X_3	6	100	14,928	20,000	10	5
X_4	6	500	1,190	20,000	50	5
X_5	6	100	34,872	40,000	10	5
X_6	6	200	14,504	40,000	50	5
X_7	6	4,000	15,273	40,000	50	5
X_8	3	100	44,950	50,000	10	5
X_9	3	1,000	4,871	50,000	100	5
X_10	3	5,000	4,615	50,000	100	5

1) Qualification phase: In this phase, each algorithm was executed with instances of Set A. Here, the 30 best scores of each category where selected.
2) Final phase: Each algorithm was executed with large sized instances (groups B and X), from 5,000 to 50,000 processes and 100 to 5,000 machines.

III. REVIEW OF APPROACHES

This section presents a review of recently proposed approaches for the Google Machine Reassignment Problem (GMRP). For each algorithm, we present a summary of their algorithmic features, details of their main components, the instances used to evaluate the algorithm and the performance of the proposed algorithm. Considering the complexity of the GMRP, we present an analysis of how the problem is faced by each strategy, in terms of the satisfaction of the constraints and the optimization.

As we mentioned, for the competition, some experimental setup rules were defined: (1) execution time limit of 300 seconds, (2) algorithms are evaluated sequentially. Here, we divide the algorithms in two main groups: approaches that were evaluated using the *Same Experimental Setup* of the competition or a *Different Experimental Setup* (e.g., execution time, executed in parallel, among others). To analyze the performance of each algorithm, we present the number of instances that each technique reaches the actual best known solution for each set. For this, we considered the best quality solution obtained by each algorithm. A discussion and further analysis related to the performance of each algorithm is presented in the next section.

A. SAME EXPERIMENTAL SETUP

A Variable Neighborhood Search (VNS) algorithm is presented in [7]. This algorithm is a local search algorithm focused on the reduction of the Load Cost of the objective function. The design of VNS is mostly based on the requirements of each process, in terms of resources.[3] As a consequence, VNS is mostly focused on the satisfiability of resource-related constraints (Capacity and Transient Usage). During its search process, VNS tries to obtain better quality candidate solutions using four different operators to produce feasible solutions:

- *Shift*: move one process to another machine,
- *Swap*: interchanges two processes from different machines,
- *Chain*: shift a defined number of processes at the same time,
- *Big Process Rearrangement (BPR)*: reassign big processes to a target machine. For this, other smaller processes could be also reassigned to generate enough space.

Authors note that when convergence is reached, it is difficult to reassign processes that consume several resources. Moreover, they suggest to reassign big processes during early stages of its search process. For this, the algorithm give priority to big processes to be reassigned first and also, operators are applied in a particular order: (1) BRP, (2) Shift, (3) Swap, and (4) Chain operator.

In order to increase the exploration of the algorithm, the objective function is modified from the original weighted sum. VNS was proposed for the ROADEF challenge and obtained the first place during the final phase of the competition. To evaluate VNS, authors performed 100 independent runs considering instances from sets A and B. However, results of VNS on Set X are reported in [8]. Results showed that VNS reached one actual best known solutions in set A and 2 in set X.

Two Large Neighborhood Search (LNS) approaches are presented in [9], one based on a Mixed Integer Linear Programming model (LNS-MIP) and the other based on a Constraint Programming (LNS-CP) model. Both algorithms are collaborative techniques, where Local Search and Exhaustive

[3] Authors define Big Processes and Small processes, related to the amount of resources consumed for their assignation.

Search strategies cooperate to solve the GMRP. CPLEX is used in both techniques.[4] In LNS-MIP, the algorithm creates and solves sub-problems iteratively. For this, from a selected subset of machines, their processes are reassigned until a time limit is reached. On the other hand, the LNS-CP is divided in two main phases: (1) select and create a sub-problem (where a subset of processes and machines are selected), and (2) optimize the sub-problem. In the second phase, a Systematic Search is applied with a stopping criteria based on the number of failures and a given threshold. For the Systematic Search, three components are used: variable ordering heuristics, value ordering heuristics and filtering rules applying constraint propagation. Constraint satisfaction is mostly ensured on the first phase and on the Constraint Propagation. On the other hand, the second phase is focused in improving the quality of candidate solutions. Instances from set A were used to evaluate LNS-MIP and instances from sets A and B to evaluate LNS-CP. However, results of LNS-CP on Set X are reported in [8]. Results showed that both algorithms reached one actual best known solution on set A.

A local search based algorithm named Multi-Start Iterated Local Search (MS-ILS) was presented in [10]. MS-ILS was proposed for solving the Multi-Capacity Bin Packing Problems (MCBPP) and the GMRP. Based on Iterated Local Search (ILS) [11], the algorithm includes a local search procedure that considers Swap and Shift operators. First, a subset of machines $\bar{M}$ is constructed optimizing only the Load and Balance costs. Then, both operators are independently applied considering a machine m_s selected from $\bar{M}$. The operator that obtains the higher quality solution is accepted. In order to balance the exploitation and exploration on MS-ILS, the local search procedure can be applied to the best solution found or to the current solution in each iteration.

MS-ILS includes a multi-start component to escape from local optima solutions. For this, two operators named *shaking moves* were used to produce feasible solutions:

- *Home Relocate*: selects randomly k processes that are currently not hosted on their initial machine, and relocates them to their initial machine.
- *K-Swap*: randomly selects k times a pair of machines, and performs a swap of random groups of either 3, 4, or 5 processes, with equal probability, among these machines.

With the objective of decreasing the execution time of MS-ILS, the shaking moves are triggered dynamically when the improvement of the objective functions is less than a certain threshold. The algorithm was evaluated using sets A, B and X. Results showed that MS-ILS obtained one actual best known solution on set A.

A hyperheuristic based in Simulated Annealing named HH-MRP was proposed in [12]. HH-MRP uses two low-level heuristics (*H0* and *H1*) and each one applies some operators to candidate solutions:

- *H0*: this heuristic applies *Shift*, *Swap* and *Backtracking* (perform a backtracking process to a subproblem) operators.
- *H1*: this heuristic applies *Shift*, *Swap* and *Double Shift* (tries to perform Shift operator re-assigning other processes if it is necessary) operators.

In both heuristics, the decision of which operators are applied is stochastically performed using some probability parameters. In order to produce a balance between exploration and exploitation of the search space, HH-MRP uses a self-adaptive strategy. The general schema of HH-MRP are three phases. On the first phase, *H0* is applied for a fixed time limit and the temperature is reduced iteratively. Then, the second phase independently applies *H0* and *H1* to the current solution during a specific amount of time (maintaining the temperature). The obtained solution with the higher quality will be used on the next phase. In the third phase, the selected heuristic is executed again, during a higher amount of time in order to improve the obtained solution. These phases are repeated until the execution time limit is reached. Parameter values were defined using a tuner algorithm named ParamILS [13]. To evaluate the performance of HH-MRP, instances from sets B and X were used. Results showed that HH-MRP is competitive to the three best teams of the Senior category of the challenge. However, none actual best known solution are reached.

In [14], an analysis of the GMRP from a Vector Bin Packing perspective (VBP) is presented (H-VBP). Authors propose a generalization of VBP that allows distinct capacities of bins for each dimension called Vector Bin Packing with Heterogeneous Bins (VBPHB) (resulting the same multi-capacity bin packing referred by [10]). Here, authors used several greedy-based heuristics to construct solutions for VBPHB classified into: item centric heuristics, bin centric heuristics and bin balancing heuristics. Then, an analysis of some structural properties of the MRP are presented. Authors conclude that instances can be decomposed into smaller sub-problems and showed that the mentioned heuristics can be adapted to the MRP. Based on this, a multi-start algorithm was proposed to generate feasible solutions assigning all processes (instead of using the provided initial solution s_0). Even though the MRP is a very constrained problem, the proposed heuristics were able to assign more than the 90% of the processes in all the instances. Authors denote that the best heuristics, in terms of the number of feasible solutions, were the item centric heuristics with priorities on processes and machines (randomly ordered or normalized by bins capacities). Moreover, using item centric heuristics, feasible solutions were obtained in 1 instance of the set A, 5 in set B and 6 in set X. Using bin centric heuristics, feasible solutions were obtained in 2 instances from set A, 4 from set B and 5 from set X. About bin balancing heuristics, feasible solutions were obtained in 1 instance from set A, 5 from set B and 4 from set X.

A hybrid approach based on Hill Climibing and Large Neighborhood Search (LNS) was presented in [15].

[4] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

The algorithm HC-LNSHC has two main components, a First Improvement Hill Climbing (FI-HC) algorithm and a LNS algorithm (which uses Mixed Integer Programming to solve sub-problems). Starting from the provided initial solution, it executes FI-HC to improve the quality of solution using a Shift operator. The idea is to reassign processes to produce feasible candidate solutions until a local optima is reached. For this, a *process potential* preprocessing is executed, to apply the Shift operator giving priority to some processes that can produce higher reductions of the Load and Move costs. Moreover, a Tabu List is used to consider only machines that produce an improvement of the candidate solution.

Then, LNS is applied to perform additional improvements creating sub-problems that consider a subset of the total machines. The LNS component tries to move several processes at the same time and CPLEX solver is used to solve each sub-problem. Authors conclude that the use of FI-HC showed improvements mainly in the large size instances. Moreover, authors note that considering the large number of constraints and dependencies of the instances of the GMRP, re-assign many processes at once is indispensable to obtain high quality results. Results in instances of sets A and B showed that HC-LNSHC reached one best known solution on set B.

A hybrid approach based in Iterated Local Search and Integer Programming (IP) is presented in [16]. IP-RILS algorithm considers specialized versions of the Shift and Swap operators. *Shift Machine Load Sort* is an implementation of Shift that considers only a subset of machines previously sorted. For this, processes of machines with higher values of Load Cost are first re-assigned. The idea is to optimize the quality of candidate solutions focused on reducing the Load Cost. On the same line, authors consider that processes that demand more resources are harder to be re-assigned. *Swap in Service* operator is a variation of the typical Swap, considering pairs of processes of machines only from the same services. The idea is to improve the quality of candidate solutions, performing Swap perturbations without violate dependency, spread, and conflict constraints. IP-RILS uses an *IP-perturbation* component, to solve sub-problems using IP and Branch and Bound. To evaluate IP-RILS, instances of sets B and X were used. Results on instances from sets B and X showed that IP-RILS reached zero best known solutions.

A Multi-Neighborhood Local Search (MNLS) approach was proposed in [17]. Starting from the initial solution provided on the instances from the challenge, three operators are sequentially used in order to explore different neighborhood structures: Shift, Swap and (three processes) Swap. As the instances consider several machines and processes, a *neighborhood partition technique* is considered on MNLS in order to reduce the execution time of the algorithm. As a consequence, the operators are applied considering specific parts of the neighborhood. During the local search phase, unfeasible moves are accepted. For this, a repair strategy is included that re-assigns processes when a machine violates capacity or transient usage constraints, until feasibility is recovered. After the local search procedure, a perturbation operator is performed to escape from local optimum solutions. Here, one randomly selected operator (from the three previously mentioned) is applied to produce a feasible candidate solution and continue the search process. To evaluate the performance of MNLS, instances from sets A, B and X were used. Results showed that MNLS reached one actual best known solution on set A.

In [18], an optimization-based heuristic technique was proposed (OBH-S27). This technique requires the decomposition of the problem into a sequence of small-sized instances, that are iteratively solved using a general Mixed Integer Programming (MIP) solver. To create the sub-problems, a subset of μ machines is considered. The most loaded $\mu/2$ machines and also, some lightly loaded machines are selected. The objective is to balance the load excess in some machines. Starting from the initial assignment s_0, four procedures are used to improve assignments in a sub-problem: (1) Inter-neighborhood procedure (reassign processes considering a subset of machines), (2) Intra-neighborhood procedure (reassign processes considering machines of the same neighborhood), (3) Swap considering a sub-problem and (4) Intra-service procedure (reassign processes from machines of the same service). C-PLEX 12.4 is used for solving each sub-problem. Authors denote that as the load cost largely dominates the other cost components, OBH-S27 consider a relaxation of the objective function, restricted to the load cost. To evaluate the performance of OBH-S27, experiments were performed considering instances of sets A, B and X. Results showed that OBH-S27 reached one actual best known solution in set A.

A neighborhood analysis is presented in [19], studying the effect of three different operators. The objective is to analyze the effect of performing a search process using *Swap*, *Shift* or *MoveBig* (a large process is selected and moved) to generate feasible candidate solutions. Each operator is coupled to a Steepest Descent (SD) local search algorithm. SD is a parameter-free algorithm that repeatedly replace the current solution by the best newly generated solution until there is no more improvement. Also, SD-Combined is studied, where any of these three operators is randomly selected and applied in each step of the SD algorithm. Results in set A instances showed that SD-Combined obtained the best results and also, reached one actual best known solution. Authors conclude that generating neighbors based on moving blocks of processes seems more effective than that based on moving single processes.

B. DIFFERENT EXPERIMENTAL SETUP

A Simulated Annealing (SA-GMRP) algorithm was proposed in [20]. SA-GMRP considers two operators to perturb candidate solutions: *Shift* and *Swap*. The decision of which operator will be applied is stochastically performed using a probability parameter p. Considering that neighborhoods can contain several candidate solutions, authors applied a sampling strategy to reduce the execution time and also, evaluate

the feasibility of fewer solutions. For the *Shift* operator, considering a randomly selected machine m, a given process is re-assigned to neighbor machines of m. On the *Swap* operator, considering a machine and a process randomly selected, the re-assignation is performed considering a second process chosen from a particularly ordered list of processes. The temperature value of SA-GMRP is decreased geometrically by a factor r. With the objective of applying more significant perturbations and to escape from local optima solutions, a reheating procedure is included in SA-GMRP. Here, when the rate of accepted moves is lower than 0.01%, the solution is considered ''frozen'' and the temperature is incremented in $\frac{t_0}{100}$ (where t_0 is the initial temperature). SA-GMRP was executed in a dual-core machine and evaluated considering instances from sets A and B. Results showed that SA-GMRP obtained one current best known solution in the A set.

Some improvements to LNS-CP were presented in [21]. First, considering that LNS requires the definition of some parameter values (size of the neighborhood, when to change the neighborhood size, among others), a parameter analysis is presented. A two phase model for solving instances was proposed ($LNS - CP^*$). First, an offline phase is performed, which consists in a clusterization process of instances and a parameter tuning process per cluster. Instances were grouped considering their problem definition features: number of machines, processes, resources, among others. Then, during an online phase, a new incoming instance is solved using the values for the parameters of the closest cluster. Two algorithms were used for the parameter tuning process: Gender-Based Genetic Algorithm (GGA) [22] and Instance-Specific Algorithm Configuration (ISAC) [23]. Additionally, $LNS - CP^*$ algorithm was parallelized and evaluated using a time limit of 100 seconds and different number of cores. To evaluate the algorithm, authors generate 1245 instances based on instances of set B. Also, a subset of these instances was used for the clusterization process on the offline phase. Results on instances from set B show that using the obtained parameter configurations produce improvements of a 19%, 31% and 42% using 1, 2 and 4 cores, respectively. Authors confirm the importance of consume an amount of resources defining a suitable set of parameter values for $LNS - CP^*$.

Noisy Local Search (NLS) is presented in [24]. Similarly to VNS, NLS is a multi-start local search algorithm that applies three operators: *Swap*, *Shift* and *BPR* (Big Process Reassignment). These operators are applied in a Round-Robin manner to obtain feasible candidate solutions. Previously to the execution of NLS, a preprocessing step is performed to sort the processes considering their size (the sum of its requirements over all resources). Larger processes are re-assigned first and a adaptive-size subset of processes is considered to be reassigned. Authors present a detailed analysis of the importance of using the BPR operator to improve the quality of the obtained solutions in a subset of instances ($A2\_2$, $A2\_3$, $A2\_5$, B_1 and X_1). However, the usage of BPR operator increases the execution time of NLS between 3 to 7 times. During the BPR operator, all constraints except capacity constraint (conflict, spread, dependency) are satisfied at any time. A perturbation performed by BPR operator is accepted when all constraints are satisfied and the solution cost has been improved. To escape of local optima solutions a noising strategy is included on NLS. For this, a modification of the objective function is performed, increasing the load cost weights for one or a subset of resources. NLS also includes a mechanism to deal with random seeds. On the initial phase of the search, the biggest processes are reassigned considering different seeds. Then, considering the best seeds, the search process is continued. To evaluate NLS, 100 independent runs were performed considering instances from the three sets of instances. Results showed that NLS reached 2 best known solutions on the A set, zero on set B and 2 on set X.

Fast Machine Reassignment (FMR), a cooperative search approach, was proposed in [25]. FMR is a parallel technique that in each thread a set of different approaches can be executed. The communication between threads only allows to share the best known solution and the number of exchanges is controlled. The first thread can execute the following approaches:

- Greedy heuristic, to build alternate feasible initial solutions different from s_0,
- Shift operator,
- Ejection Chain operator, that tries to apply swap operators that minimize the total cost. Particularly, the idea is to replace processes assignments, iteratively selecting destination machines that produce a decrement in the total cost,
- Adaptive Variable Neighborhood Search, that dynamically changes the operator to be applied (swap, shift or ejection chain) using a roulette wheel selection strategy (based in a score per operator)

On the other hand, the second thread can apply:

- Extended Best Shift, that performs a shift moving a process to the ''best'' available machine (which reduces the total cost the most),
- Ejection Chain operator,
- Simulated Annealing Hyperheuristic, that combines heuristic methods and acceptance criteria. The selection strategies are Temperature Descent (using typical Simulated Annealing acceptance criteria) and Fast Minimum (accepts only improvements). Shift or swap operators are used to perturb solutions.

To evaluate FMR, authors performed a comparison of using the cooperative scheme or using both threads separately. Results showed that the cooperative scheme outperforms the independent approach. Moreover, results in sets B and X showed that FMR obtained close results to the best team of the competition.

A constraint-based Large Neighborhood Search (CBLNS-J25) was proposed in [26]. This algorithm is a LNS that iteratively works with a sub-problem that consider a subset of processes to be reassigned. Each sub-problem is optimized using Constraint Programming as a depth first

search. To explore different neighborhoods, different search strategies can be applied:

- Random Search (process are reassigned randomly chosen with a selection probability proportional to its contribution to the overall cost function),
- Process Neighborhood Search (most expensive processes and some randomly selected ones are reassigned),
- Target Move Search (big size processes are reassigned to machines that have enough resources) and,
- Undo Move Search (undo the reassignment of a process).

To allow expensive or big size processes to be reassigned, Process Neighborhood Search and Move Search require to move other subset of k processes. Only improvement perturbations are allowed in these strategies. In order to analyze the convergence of the algorithms, authors present an overview of how the cost of the obtained solutions considering 1, 5 and 30 minutes. Results in instances from set A, B and X showed that CBLNS-J25 is competitive to the first places of the ROADEF/EURO competition.

In [27], an evolutionary parallel Late Acceptance Hill Climbing (P-LAHC) algorithm was proposed. This algorithm is a cooperative hybrid evolutionary approach where an initial population of individuals is generated from the provided initial solution. Each initial solution is assigned to a Late Acceptance Hill Climbing (LAHC) [28], that is executed to improve the solution randomly moving processes across machines.[5] LAHC is mostly focused in optimize the total cost, maintaining the feasibility of obtained solutions. A restart mechanism is used when no improvement has been obtained during a certain number of iterations. For this, a mutation operator is applied in order to modify the solutions without improvement. These operators can be Swap, Double Swap (same as single swap but with four processes from two different machines), Shift or Double Shift (same as single move but with two processes). Results in instances from sets A and B showed that P-LAHC reached one actual best known in set A and zero in set B. P-LAHC algorithm is executed in parallel considering 4 threads.

An Evolutionary Simulated Annealing (ESA) algorithm was proposed in [29]. ESA considers a population of candidate solutions, where each solution has its own Simulated Annealing (SA) algorithm. Each SA algorithm is executed in parallel, starting from a different initial solution obtained randomly modifying s_0. The idea is to produce diverse solutions and as a consequence, different local optima solutions. ESA uses a mutation operator to escape from local optima solutions. Each candidate solution considers its own mutation operator, that can be one of the following operators: Swap, Double Swap, Shift and Double Shift. Results in instances from set A and B shows that ESA is competitive to state-of-the-art algorithms. Moreover, ESA reaches one actual best known in set A.

In [29], the Evolutionary Learning based Iterated Local Search (EL-ILS) was proposed. EL-ILS is focused in performing a learning process, in order to select the most suitable technique for a problem that will be tackled. EL-ILS has three main components: a meta-feature extraction, meta-learning and classification. The two first steps are focused in generate a multi-class classifier by training (considering a sub-set of problem instances). The classifier, generated by a Genetic Programming approach, selects the most suitable Iterated Local Search algorithm to be applied. The idea is to learn from the existing correlations between problem instances and search techniques considering the following characteristics: number of machines, number of resources, number of processes, number of services, number of neighborhoods, number of dependencies, number of locations, number of balance costs and number of resources needing transient usage. Each Iterated Local Search approach has its own operator and also its own perturbation operator. The possible operators can be: Swap, Double Swap, Shift and Double Shift. On the other hand, possible perturbation operators can be: random perturbation (randomly swap some processes) or guided perturbation (large size processes are reassigned). Results on a subset of problems of set A, B and X shows that EL-ILS is competitive to the actual state-of-the-art algorithms.

The Multi-neighborhood Great Deluge (MNGD) was proposed in [30]. Great Deluge is a single point algorithm that uses the following acceptance criteria: (1) improvements are always accepted and (2) non-improvement solutions are accepted considering a defined threshold (the quality of s_0 and a small amount ϵ subtracted in each iteration). To perturb solutions, four operators are used: single swap, double swap, single shift and double shift. Results in the three benchmark instances set show that MNGD reached one best known solution in set A and one in set B.

Turky *et al.* proposed a strategy similar to P-LAHC named CHSA, a cooperative hybrid Memetic Algorithm coupled with Simulated Annealing (SA) [8]. CHSA is a population-based algorithm that was designed to be executed in parallel. Here, the evolution, perturbation and improvement of each candidate solution is executed in a thread. The generation of the initial population and the restart mechanism is equally defined as in [27]. Then, each initial solution is assigned to a Simulated Annealing algorithm and each one is executed in parallel with a specifically defined configuration: a perturbation operator, an initial temperature value and a cooling coefficient. Possible operators are: single swap, double swap, single shift and double shift. A candidate solution is accepted if its quality is better than the current solution or considering a probabilistic decision. The probability P of acceptance of new candidate solutions is defined as:

$$P = \exp^{\frac{-(f(x')-f(x))}{t}} \tag{2}$$

where x is the current candidate solution, x' is the new candidate solution and t is the current temperature value. To ensure

[5] LAHC is a Hill Climbing algorithm that its acceptance criteria is particularly defined. The algorithm contains a list L of the solutions generated in the last iterations. A newly generated solution will be accepted if its quality: (1) its higher than the quality of the current solution or, (2) its higher than the quality of a solution on L.

TABLE 6. Experimental setup conditions used in all the reviewed approaches-N.P: Not provided.

Algorithm	Experimental Setup	# Seeds	Execution time	Parallel	Conversion Factor
Competition Configuration	Intel Core 2 Duo E8500 (3.16GHz, 64bits, 4GB of RAM)	1	300 seconds	No	-
VNS [7]	Intel Core i7-920 (2.66 GHz, 64bits, 6GB of RAM)	100	300 seconds	No	0.95
LNS-CP, LNS-MIP [9]	N.P	N.P	300 seconds	No	-
SA-GMRP [20]	N.P	N.P	300 seconds	Yes	-
MS-ILS [10]	AMD Opteron 6378 (2.4 GHz, 64bits, 4 GB of RAM)	40	300 seconds	No	0.85
$LNS-CP^*$ [21]	Intel Dual Quad Core Xeon (2.66GHz, 64 bits, 12 GB of RAM)	N.P	100 seconds	Yes	2.50
NLS [24]	Intel Core 2 Duo E8500 (3.16 GHz, 64 bits, 4 GB of RAM)	100	300 seconds	Yes	1.00
IP-RILS [16]	Intel Core 2 Duo E8400 (3.0 GHz, 64 bits, 4 GB of RAM)	25	300 seconds	No	0.99
HH-MRP [12]	AMD Quad Core FX-4300 (3.8 GHz, 64 bits, 4GB of RAM)	5	300 seconds	No	1.13
HC-LNSHC [15]	Intel Core i7 950 (3.07 GHz, 64 bits, 6GB of RAM)	25	300 seconds	No	1.06
CBLNS-J25 [26]	12-core AMD Opteron 6172 (2.1 GHz, 64bits, 256GB of RAM)	30	300 seconds	Yes	4.35
FMR [25]	Intel Core i7-2600 CPU (3.40GHz, 64bits)	1	300 seconds	Yes	4.43
OBH-S25 [18]	Intel Core 2 Duo E8500 (3.16GHz, 64bits, 4GB of RAM)	1	300 seconds	No	1.00
	Intel Core Quad i7-2600 (3.4Ghz, 64bits, 16GB of RAM) - Only in instance A2_4				1.35
H-VBP [14]	N.P	50	N.P	No	-
P-LAHC [27]	N.P	31	300 seconds	No	-
MNLS [17]	Intel Xeon E5-2609 (2.5 GHz, 64 bits, 32 GB of RAM)	100	300 seconds	No	1.00
SD-Combined [19]	N.P	30	300 seconds	No	-
ESA [29]	N.P	31	300 seconds	Yes	-
EL-ILS [33]	N.P	31	300 seconds	No	-
MNGD [30]	N.P	31	300 seconds	No	-
CHSA [8]	N.P	31	300 seconds	Yes	-
HH [31]	N.P	31	300 seconds	No	-
Ant-HH [32]	N.P	31	300 seconds	No	-

the cooperation between threads, at each generation, the solutions of all the SAs will be compared and the best one will be saved. Moreover, to provide a new initial solution for the restart procedure, the best solution obtained from all the SA approaches is used and modified by a particularly assigned mutation operator. Considering the problem instances from the challenge, CHSA obtained one best known results in set A, one in set B and one in set X. CHSA was executed in parallel considering 4 threads.

An evolutionary hyperheuristic local search approach was proposed in [31]. Authors conclude that no single local search algorithm can consistently perform well in all the instances and also, considering a particular parameter configuration. The proposed algorithm (HH) considers a two-stage structure: the first stage (HH-LS) is focused in selecting a local search method (Simulated Annealing, Iterated Local Search, Late Acceptance Hill Climbing, Great Deluge and Steepest Descent) and the second stage (HH-OP) works with a set of neighborhood operators (single Swap, double Swap, single Shift, double Shift, Swap-Shift, Shift-Swap, BPR and Swap-BPR). HH uses a population of solutions, randomly initialized, to store the best solutions reached and to obtain diverse solutions. To select the local search method in HH-LS and to select a set of operators in HH-OP a roulette wheel mechanism is used. In [32], a modified version of HH is presented. Here, to select a local search method, a Multi-Armed Bandit (MAB) mechanism is used. Moreover, a learning automation reinforcement learning approach is used in HH-OP to select operators. The strategy defines a probability distribution for all the operators and then, a roulette wheel selection mechanism is used for selecting which operators will be used. The available operators to be selected are stored in a list, controlled by a ranking mechanism. This mechanism is based on the diversity of solutions obtained during the execution of HH-LS and HH-OP. Moreover, the mechanism controls the size and diversity of the pool of operators. To update the population, a bi-objective distance-based strategy is used. For this, the quality (measured by the cost function) and a distance metric (measured by the number of different assignations) are considered to compare pairs of solutions. Using Pareto dominance, non-dominated solutions are part of the population of solutions.

To evaluate the performance of HH, authors performed 31 independent runs considering instances from sets A, B and X. Results showed that HH reached 3 best known solutions in set A and 2 in set B.

A bi-level hyperheuristic approach based in Ant Colony Optimization was presented in [32]. Ant-HH is a modified version of HH that considers several improvements. Similarly as in HH, Ant-HH is a bi-level technique: the upper-level works with the local search methods and the lower-level worked with operators. Here, an Ant Colony Optimization algorithm manages both levels, combining the most appropriate local search algorithm with sequences of operators. Ants

TABLE 7. Best known solution per instance considering the same experimental setup of the competition.

Instance	BK	Algorithms
$A1\_1$	44 306 501	VNS [7], LNS-CP [9], LNS-MIP [9], MS-ILS [10], MNLS [17], SD-Combined [19], OBH-S25 [18]
$A1\_2$	777 532 896	OBH-S25 [18]
$A1\_3$	583 005 717	MNLS [17], OBH-S25 [18]
$A1\_4$	247 433 844	HC-LNSHC [15]
$A1\_5$	727 578 309	MNLS [17], OBH-S25 [18]
$A2\_1$	167	MS-ILS [10]
$A2\_2$	720 671 548	VNS [7]
$A2\_3$	1 190 713 414	VNS [7]
$A2\_4$	1 680 609 218	OBH-S25 [18]
$A2\_5$	309 714 522	VNS [7]
B_1	3 307 124 603	VNS [7]
B_2	1 015 517 386	VNS [7]
B_3	156 519 816	HC-LNSHC [15]
B_4	4 677 817 475	LNS-CP [9]
B_5	923 312 207	HC-LNSHC [15]
B_6	9 525 421 332	HC-LNSHC [15]
B_7	14 835 031 813	VNS [7]
B_8	1 214 086 286	HC-LNSHC [15]
B_9	15 885 524 407	HC-LNSHC [15]
B_10	18 048 499 616	VNS [7]
X_1	3 030 246 091	VNS [7]
X_2	1 010 050 981	MNLS [17]
X_3	259 656	VNS [7]
X_4	4 721 727 496	MNLS [17]
X_5	144 768	VNS [7]
X_6	9 546 958 474	IP-RILS [16]
X_7	14 253 133 805	VNS [7]
X_8	83 711	MNLS [17]
X_9	16 125 675 266	MNLS [17]
X_10	17 815 045 320	VNS [7]

perform walks in a bi-level weighted graph structure which nodes are the local search methods (in first level) and the operators (in the second level). The weights represents the desirability of moving from one node to another. Pheromone is deposited in edges related to non-dominated solutions considering quality and diversity as the objectives to improve. The heuristic knowledge gives preference to pairs of local search methods (or operators) that improve the convergence of the algorithm. As in HH, Ant-HH uses a population of high quality and diverse solutions. Iteratively, each ant selects one candidate solution s_k from the population. Then, the local search methods and the set of operators is modified in s_k, considering pheromone and heuristic knowledge information. The population is updated considering the same distance-based strategy from HH. Results in sets A, B and X show that Ant-HH is one of the best algorithms of the state of the art. Ant-HH reached 9 actual best known solutions in set A, 6 in set B and 8 in set X. Experiments considered a time limit of 300 seconds and 31 independent runs.

IV. DISCUSSION

This section presents an analysis and discussion of the performance of the reviewed algorithms. The idea is to provide useful information to new researchers that are interested in evaluate and compare their approaches to the state-of-the-art algorithms. As we mentioned, the ROADEF/EURO

TABLE 8. Algorithms that reach each best known quality solution per instance.

Instances	BK	Algorithms
$A1\_1$	44 306 501	VNS [7], NLS [24], LNS-CP [9], SA-GMRP [20], LNS-MIP [9],MS-ILS [10], P-LAHC [27], OBH-S25 [18], MNLS [17], OBH-S25 [18], SD-Combined [19], MNGD [30], ESA [29] CHSA [8], HH [31], Ant-HH [32]
$A1\_2$	777 532 177	HH [31],Ant-HH [32]
$A1\_3$	583 005 715	Ant-HH [32]
$A1\_4$	244 875 200	HH [31],Ant-HH [32]
$A1\_5$	727 578 306	Ant-HH [32]
$A2\_1$	161	Ant-HH [32]
$A2\_2$	720 671 511	Ant-HH [32]
$A2\_3$	1 182 260 491	NLS [24]
$A2\_4$	1 680 368 560	Ant-HH [32]
$A2\_5$	307 150 821	Ant-HH [32]
B_1	3 291 069 365	Ant-HH [32]
B_2	1 010 949 451	MNGD [30]
B_3	156 519 816	HC-LNSHC [15]
B_4	4 677 792 536	Ant-HH [32]
B_5	922 944 510	CHSA [8], Ant-HH [32]
B_6	9 525 851 389	Ant-HH [32]
B_7	14 834 456 020	HH [31]
B_8	1 214 291 129	HH [31]
B_9	15 885 437 252	Ant-HH [32]
B_10	18 048 187 105	Ant-HH [32]
X_1	3 030 246 091	VNS [7], NLS [24]
X_2	1 002 379 317	CHSA [8], Ant-HH [32]
X_3	75 154	Ant-HH [32]
X_4	4 721 586 142	Ant-HH [32]
X_5	57 973	Ant-HH [32]
X_6	9546936159	Ant-HH [32]
X_7	14 252 476 500	Ant-HH [32]
X_8	32 014	Ant-HH [32]
X_9	16 125 531 142	Ant-HH [32]
X_10	17 815 045 320	VNS [7], NLS [24]

Challenge 2012 defines some rules for the execution of the proposed approaches. These rules are:

- Algorithms were executed sequentially
- An execution time limit of 300 seconds was considered
- Approaches were executed in an Intel Core 2 Duo E8500 (3.16GHz, 64bits, 4GB of RAM) machine.

However, different experimental setup and machine configurations were used to evaluate the reviewed approaches. Table 6 presents the experimental setup of all the reviewed approaches. For each approach, we provide details of the hardware used to evaluate each algorithm, the number of seeds used (independent executions), if it was executed in parallel and a conversion factor. This factor is used to obtain an execution time conversion between using a machine of the same characteristics that the one used on the challenge with each machine used to evaluate the reviewed approaches. For example, considering a set of routines that are executed in the Intel Core 2 Duo E8500 on 300 seconds, an Intel i7 920 (used to evaluate VNS [7]) needs $300/0.89 = 337$ seconds to perform the same set of routines.[6] Notice that in some cases,

[6]This conversion factor is obtained using information available in http://www.cpubenchmark.net.

TABLE 9. Classification of GMRP algorithms - Local Search (LS): (A)*Only LS* or (B) *Collaborative approach* - Use of preprocessing step: (C) Yes or (D) No - Weighted sum: (E) Original or (F) Modified - Experimental setup conditions: (G) Same of the competition or (H) Different scenario.

Algorithms	Local Search (LS)		Use of Preprocessing		Weighted Sum		Experimental Setup Conditions	
	(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)
VNS [7]	X		X			X	X	
LNS-CP, LNS-MIP [9]		X		X	X		X	
SA-GMRP [20]	X			X	X			X
MS-ILS [10]	X		X		X		X	
$LNS-CP^*$ [21]		X		X	X			X
NLS [24]	X		X			X		X
IP-RILS [16]		X	X		X		X	
HH-MRP [12]	X			X	X		X	
HC-LNSHC [15]		X	X		X		X	
OBH-S25 [18]		X	X			X	X	
H-VBP [14]	X		X		X		X	
FMR [25]	X		X		X			X
CBLNS-J25 [26]		X	X		X			X
P-LAHC [27]	X			X	X		X	
MNLS [17]	X		X		X		X	
SD-Combined [19]	X		X		X			
ESA [29]	X			X	X			X
EL-ILS [33]	X		X		X			X
MNGD [30]	X			X	X			X
CHSA [8]	X			X	X			X
HH [31]	X		X		X			X
Ant-HH [32]	X		X		X			X

information of the experimental setup was not provided on some related articles ('N.P' on Table 6).

A. BEST KNOWN SOLUTIONS AND STATE-OF-THE-ART ALGORITHMS

To the best of our knowledge, we present the best known quality solutions for each instance of the three benchmark sets A, B and X. Moreover, we present which algorithms are able to reach each solution. Table 7 shows the best known quality solution per instance considering the same experimental setup of the challenge (execution time limit and sequential execution). Here, the algorithms that obtained at least one best known quality solution are VNS [7], LNS-CP [9], LNS-MIP [9], MS-ILS [10], HC-LNSHC [15], MNLS [17], OBH-S25 [18], SD-Combined [19] and IP-RILS [16]. The algorithms that obtained the highest number of best known solutions are VNS [7] (12 instances), followed by MNLS [17] (7 instances) and HC-LNSHC [15] (6 instances).

Table 8 shows the best known quality solution per instance considering any experimental setup. Here, in most cases, the total cost for each instance is lower than when the experimental setup of the challenge is considered. The algorithms VNS [7], SA-GMRP [20], LNS-CP [9], LNS-MIP [9], MS-ILS [10], NLS [24], HC-LNSHC [15], P-LAHC [27], MNLS [17], OBH-S25 [18], SD-Combined [19], MNGD [30], ESA [29], CHSA [8], HH [31] and Ant-HH [32] obtained at least one best known quality solution. Moreover, the algorithms that reached the highest number of best known solutions are Ant-HH [32] (23 instances), HH [31] (5 instances), NLS [24] (4 instances) and VNS [7] (3 instances).

V. CLASSIFICATION AND TAXONOMY

This section presents a classification of the reviewed techniques. We considered different features related to the design of each algorithm. First, we observed some common characteristics:

1) All the algorithms here reviewed considered local search procedures. Mostly, we observed that the operators used to perturb candidate solutions were designed to improve the quality of the obtained solutions. Moreover, in most cases, only feasible perturbations were accepted.
2) We observed a feature related to how neighborhoods of candidate solutions are constructed. As the problem instances of the GMRP consider several machines/processes, the neighborhoods are huge. In most of these algorithms, subsets of machines/processes are considered in order to reduce the execution time and to evaluate a fewer number of candidate solutions. These subsets are created considering some criteria (e.g. the total requirements of each process, remaining capacity of machines, among others) or are randomly constructed. On the other hand, when no subsets are strictly implemented, algorithms consider a First Improvement criteria to accept new candidate solutions. The objective is similar: avoid to visit and evaluate the full neighborhood of solutions.

To classify the reviewed algorithms we considered the following features:

- All the reviewed approaches consider Local Search (LS) procedures. However, there are Collaborative Approaches that also consider Exhaustive Search components for solving sub-problems. In order to analyze the type of techniques used to tackle the GMRP instances, we classified the reviewed approaches considering if they are *Only LS* or *Collaborative Approach*.
- In some approaches, a filter or a pre-processing step is performed. We observed that these steps were included

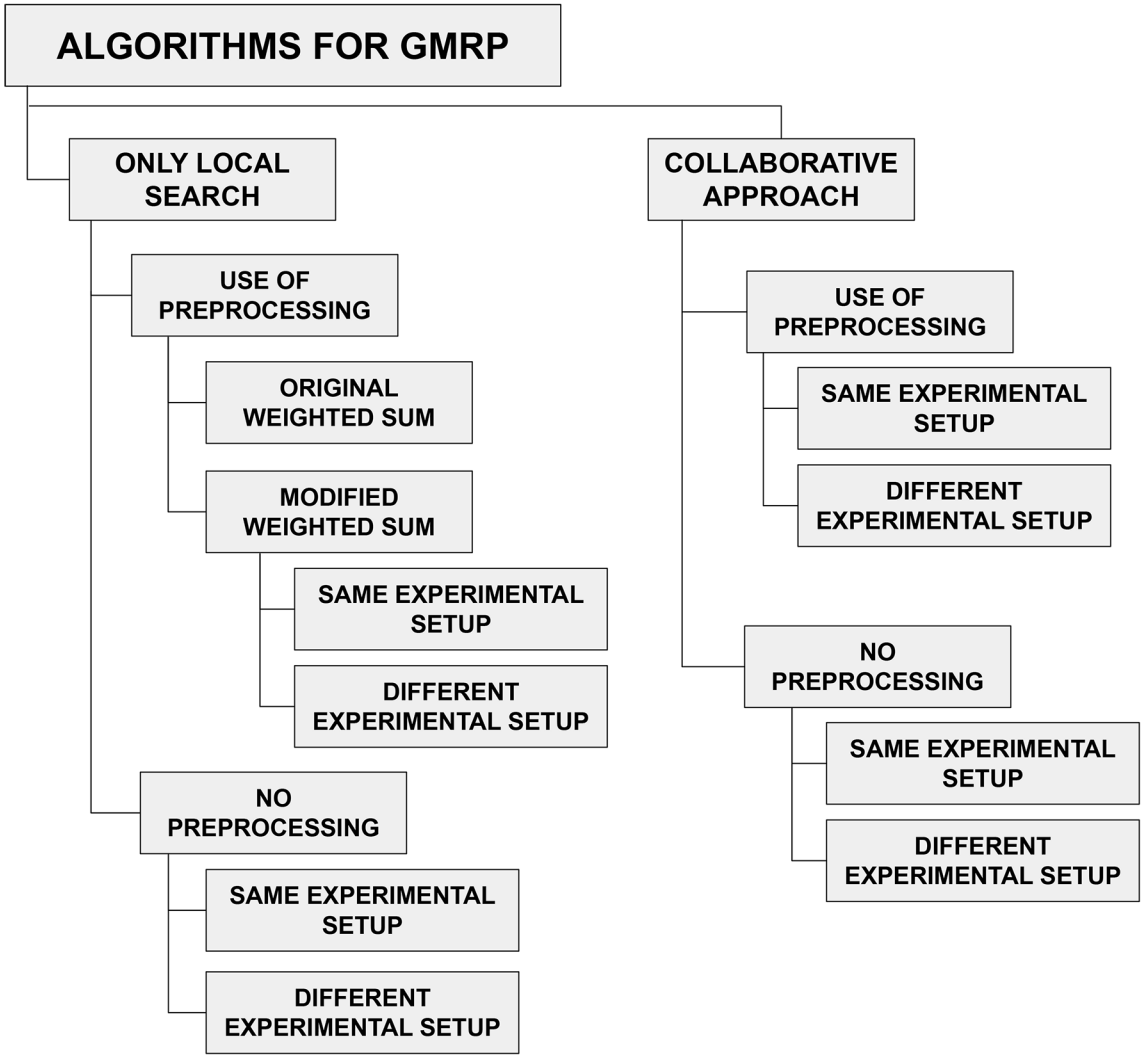

FIGURE 3. Taxonomy of the reviewed GMRP algorithms.

to discriminate between the size of processes, to construct subsets of processes, among others. For example, some approaches perform a procedure to calculate the total cost of each process, in order to discriminate which processes consume higher amounts of resources. Here, we classified the reviewed approaches considering if they *Use Preprocessing* or *No Preprocessing*. The idea is to visualize how the information provided from the problem instance is considered in the design of the reviewed approaches.

- We observed that some approaches consider a modified objective function, in order to improve some aspects of their search process. For example, some approaches give a higher priority to an specific cost modifying their weight to evaluate candidate solutions. For this, we classified the reviewed approaches in: *Original* Weighted Sum or *Modified* Weighted sum.
- We classified the reviewed algorithms in which were evaluated the *Same Experimental Setup* of the challenge or a *Different Experimental Setup* were used. The idea is to visualize the algorithmic design trend to solve this complex problem, considering different experimental scenarios and different use of the hardware resources.

Table 9 presents the classification of the reviewed techniques. The classification shows that most of these approaches are local search techniques (without using Exhaustive Search). Moreover, to tackle the GMRP, most designers consider the use of a pre-processing or a filter strategy and the original weighted sum to evaluate their candidate solutions. Finally, most algorithms were evaluated using a different experimental setup to the rules defined on the challenge. We propose a taxonomy considering the same features we used on the classification of the reviewed approaches. Figure 3 presents the proposed taxonomy.

VI. CONCLUSION

This survey presents an overview of the latest advances in solving the Google Machine Reassignment Problem (GMRP). To review these approaches we mostly considered algorithmic features, presenting their components, the

objective of each one and the performance of each algorithm. The increasing number of papers published about the GMRP, during the last years in journal and international conferences, makes very difficult the task to clearly understand the existing approaches. For new interested researchers, in order to make easier the evaluation of their algorithms, we presented an updated list of the best known solutions for the three most used benchmark instances of this problem. Moreover, we reported an updated list of the state-of-the-art algorithms for each instance of these three sets of instances in two scenarios: considering the same experimental setup of the ROADEF/EURO challenge 2012 or considering any experimental setup.

We presented a classification and a taxonomy of the reviewed approaches. First, we observed some similar features related to how the search process is performed: (1) in order to reduce the execution time of each algorithm, subset of processes or first-improvement criteria were considered on the local search procedures and (2) operators were mostly designed to improve the quality of the candidate solutions. On the other hand, we classified the approaches considering which type of technique is used to tackle the GMRP (only Local Search or a Collaborative approach), the design of the strategy to tackle the GMRP and how the evaluation function is considered during their search process. Mostly, the reviewed algorithms are pure local search techniques and also, a pre-processing step is used to (mostly) discriminate different types of processes. This situation shows that algorithms are designed considering the difficulty and the size of some problem instances of the GMRP. Also, most of the reviewed algorithms were evaluated considering a experimental setup different to the one used on the challenge and also, most approaches use the original weighted sum to evaluate the obtained candidate solutions.

Comparing the design of strategies, we observed that recently proposed approaches are algorithms that are focused in selecting suitable techniques to be applied. As GMRP instances have different features and characteristics, it is impossible to obtain one specifically designed strategy to solve all the problem instances. The newest approaches are hyperheuristics that manage existing operators, heuristics and techniques to solve GMRP instances. Moreover, both techniques (HH and Ant-HH) are the ones that reach the highest number of actual best known solutions and are the state-of-the-art for the GMRP.

Our future research will be focused on designing inexpensive methods and improving existing ones to solve GMRP. For example, techniques to deal with inter-task distribution can be promising [34]. One of the major challenge is to be able to solve large problems that are part of the hardest ones.

APPENDIX A
MATHEMATICAL MODEL

In this section, we present the mathematical formulation proposed by ROADEF/EURO in [6]. We consider a set of machines $\mathcal{M}$, which have a set of resources $\mathcal{R}$ that can be used by a set of processes $\mathcal{P}$. Some of these resources need transient usage, meaning that they will remain used even after a process is removed from the corresponding machine. The set of transient resources is denoted by $\mathcal{TR} \subseteq \mathcal{R}$. The processes $p \in \mathcal{P}$ are grouped into a set of services $\mathcal{S}$, while machines $m \in \mathcal{M}$ are grouped into a set of locations $\mathcal{L}$ and a set of neighborhoods $\mathcal{N}$. $\mathcal{S}$, $\mathcal{L}$ and $\mathcal{N}$, are all disjoint sets.

A. INPUT PARAMETERS

- $C(m, r)$: Capacity on each machine $m \in \mathcal{M}$ for a resource $r \in \mathcal{R}$.
- $R(p, r)$: Resource $r \in \mathcal{R}$ capacity required by the process $p \in \mathcal{P}$.
- $SpreadMin(s)$: Minimum number of different locations $l \in \mathcal{L}$ where the processes of each service $s \in \mathcal{S}$ have to be distributed.
- $SC(m, r)$: Safety capacity for a resource $r \in \mathcal{R}$ on each machine $m \in \mathcal{M}$.
- $PMC(p)$: Cost of moving each process $p \in \mathcal{P}$ from its initial machine $M_0(p)$ to any other one.
- $MMC(m_0(p), m_1(p))$: Cost of moving a process $p \in \mathcal{P}$ from machine m_0 to another machine m_1.
- $target_{r_1,r_2}$: Proportional relationship between resources r_1 and r_2, belonging to a triple from the balance cost. Each triple has the form $\{Resource_1, Resource_2, N\}$, where N is a proportion between both resources.
- $weight_{loadCost}(r)$: Weight of load cost for a specific resource r.
- $weight_{balanceCost}(b)$: Weight of balance cost for a specific balance triple b.
- $weight_{processMoveCost}$: Weight of the total process move cost.
- $weight_{serviceMoveCost}$: Weight of the total service move cost.
- $weight_{machineMoveCost}$: Weight of the total machine move cost.

B. DECISION VARIABLES

- $M(p) = m$: represents the assignment of the process p to the machine m, where $M : [1, p] \rightarrow [1, m]$.
- $U(m, r)$: is the usage of resource $r \in \mathcal{R}$ on the machine $m \in \mathcal{M}$, where $U : [1, m][1, r] \rightarrow \mathbb{R}^+$. Formally:

$$U(m, r) = \sum_{p \in \mathcal{P} | M(p) = m} R(p, r) \tag{3}$$

C. CONSTRAINTS

The following constraints have to be satisfied.

- **Capacity constraints:** The resource requirement of all processes $p \in \mathcal{P}$ assigned to machine $m \in \mathcal{M}$ must not exceed the capacity of each resource on that machine, $C(m, r)$.

$$U(m, r) \leq C(m, r), \quad \forall m \in \mathcal{M},\ r \in \mathcal{R} \tag{4}$$

- **Transient usage constraints:** Ensure the capacity constraints are satisfied, considering the transient resources

that are not available on the machine.

$$\sum_{p\in\mathcal{P}|M_0(p)=m\vee M(p)=m} R(p,r) \leq C(m,r),\ \forall m\in\mathcal{M},\ r\in\mathcal{TR} \quad (5)$$

- **Conflict constraints:** Processes $p \in \mathcal{P}$ belonging to a service $s \in \mathcal{S}$ cannot be assigned to the same machine.

$$(p_i,p_j)\in s^2,\ p_i\neq p_j \Rightarrow M(p_i)\neq M(p_j),\quad \forall s\in\mathcal{S} \quad (6)$$

- **Spread constraints:** Processes $p \in \mathcal{P}$ belonging to a service $s \in \mathcal{S}$ must be assigned to at least a minimum number of different locations $l \in \mathcal{L}$, determined by the *spreadMin* value of the service.

$$\sum_{l\in\mathcal{L}} min\left(1, \left|\{p\in s \mid M(p)\in l\}\right|\right) \geq spreadMin(s),\ \forall s\in\mathcal{S} \quad (7)$$

- **Dependency constraints:** If a service $s_a \in \mathcal{S}$ depends on service $s_b \in S$, then each process of s_a must be assigned to a neighborhood $n \in \mathcal{N}$ used by a process of s_b.

$$\forall p_a\in s_a,\ \exists p_b\in s_b \wedge n\in\mathcal{N} \mid M(p_a)\in n\wedge M(p_b)\in n \quad (8)$$

D. OBJECTIVES

The objective function is composed by a set of cost functions, aiming to improve the process distribution on the machines.

- **Load cost:** Sum of the costs for exceeding the safety capacity $SC(m,r)$ of a resource $r \in \mathcal{R}$ on each machine $m \in \mathcal{M}$. The total usage of the resource $r \in \mathcal{R}$ on machine $m \in \mathcal{M}$ is $U(m,r)$.

$$loadCost(r) = \sum_{m\in\mathcal{M}} max\,(0, U(m,r) - SC(m,r)) \quad (9)$$

- **Balance cost:** Sum of the costs for using the resources of a machine unevenly, with the purpose of finding a balanced assignment, allowing the addition of new processes in the future. This is done by ensuring an available ratio of two different resources, represented as a triple $b = \{r_1, r_2, target\}$. The set of triples for all pairs of resources is $\mathcal{B}$. The available capacity of the resource $r \in \mathcal{R}$ on the machine $m \in \mathcal{M}$ is $A(m,r)$.

$$balanceCost(b) = \sum_{m\in\mathcal{M}} max\,(0, target\cdot A(m,r_1) - A(m,r_2)) \quad with\ A(m,r) = C(m,r) - U(m,r) \quad (10)$$

- **Process move cost:** Sum of the costs for moving processes from their initial machine $M_0(p)$. The cost of moving process p is $PMC(p)$.

$$processMoveCost = \sum_{p\in\mathcal{P}\setminus M(p)\neq M_0(p)} PMC(p) \quad (11)$$

- **Service cost move:** Maximum amount of moved processes among all services.

$$serviceMoveCost = \max_{s\in\mathcal{S}} \left(\left|\{p\in s\setminus M(p)\neq M_0(p)\}\right|\right) \quad (12)$$

- **Machine move cost:** Sum of the costs for moving any process between two specific machines. The cost of moving a process from m_{source} to $m_{destination}$ is $MMC(m_{source}, m_{destination})$.

$$machineMoveCost = \sum_{p\in\mathcal{P}} MMC(M_0(p), M(p)) \quad (13)$$

- **Objective function:** The total objective cost function is the minimization of the weighted sum of all the mentioned costs.

$$\begin{aligned} minimize\ totalCost &= \sum_{r\in\mathcal{R}} weight_{loadCost}(r)\cdot loadCost(r) \\ &+ \sum_{b\in\mathcal{B}} weight_{balanceCost}(b)\cdot balanceCost(b) \\ &+ weight_{processMoveCost}\cdot processMoveCost \\ &+ weight_{serviceMoveCost}\cdot serviceMoveCost \\ &+ weight_{machineMoveCost}\cdot machineMoveCost \end{aligned} \quad (14)$$

REFERENCES

[1] O. Edenhofer, *Climate Change 2014: Mitigation of Climate Change*, vol. 3. Cambridge, U.K.: Cambridge Univ. Press, 2015.
[2] United Nations. (May 2016). *Framework Convention on Climate Change*. [Online]. Available: http://bigpicture.unfccc.int/#content-the-paris-agreemen
[3] EPA: US Environmental Protection Agency. (Apr. 2017). *Sources of Greenhouse Gas Emissions*. [Online]. Available: https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions
[4] G. M. Portal, M. Ritt, L. M. Borba, and L. S. Buriol, "Simulated annealing for the machine reassignment problem," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 93–114, Jul. 2016.
[5] H. M. Afsar, C. Artigues, E. Bourreau, and S. Kedad-Sidhoum, "Machine reassignment problem: The ROADEF/EURO challenge 2012," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 1–17, Jul. 2016.
[6] *2011-2012: Machine Reassignment Problem*. Accessed: May 11, 2020. [Online]. Available: http://challenge.roadef.org/2012/en/
[7] H. Gavranović, M. Buljubašić, and E. Demirović, "Variable neighborhood search for Google machine reassignment problem," *Electron. Notes Discrete Math.*, vol. 39, pp. 209–216, Dec. 2012.
[8] A. Turky, N. R. Sabar, and A. Song, "Cooperative evolutionary heterogeneous simulated annealing algorithm for Google machine reassignment problem," *Genetic Program. Evolvable Mach.*, vol. 19, nos. 1–2, pp. 183–210, Jun. 2018.
[9] D. Mehta, B. O'Sullivan, and H. Simonis, "Comparing solution methods for the machine reassignment problem," in *Principles and Practice of Constraint Programming* (Lecture Notes in Computer Science), vol. 7514, M. Milano, Ed. Québec City, QC, Canada: Springer, Oct. 2012, pp. 782–797.
[10] R. Masson, T. Vidal, J. Michallet, P. H. V. Penna, V. Petrucci, A. Subramanian, and H. Dubedout, "An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems," *Expert Syst. Appl.*, vol. 40, no. 13, pp. 5266–5275, Oct. 2013.

[11] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," in *Handbook of Metaheuristics*. Cham, Switzerland: Springer, 2010, pp. 363–397.
[12] R. Hoffmann, M.-C. Riff, E. Montero, and N. Rojas, "Google challenge: A hyperheuristic for the machine reassignment problem," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Sendai, Japan, May 2015, pp. 846–853, doi: 10.1109/CEC.2015.7256979.
[13] F. Hutter, H. Hoos, and T. Stützle, "Automatic algorithm configuration based on local search," in *Proc. 22nd AAAI Conf. Artif. Intell.* Vancouver, BC, Canada: AAAI Press, Jul. 2007, pp. 1152–1157.
[14] M. Gabay and S. Zaourar, "Vector bin packing with heterogeneous bins: Application to the machine reassignment problem," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 161–194, Jul. 2016.
[15] W. Jaśkowski, M. Szubert, and P. Gawron, "A hybrid MIP-based large neighborhood search heuristic for solving the machine reassignment problem," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 33–62, Jul. 2016.
[16] R. Lopes, V. W. C. Morais, T. F. Noronha, and V. A. A. Souza, "Heuristics and matheuristics for a real-life machine reassignment problem," *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 77–95, Jan. 2015.
[17] Z. Wang, Z. Lü, and T. Ye, "Multi-neighborhood local search optimization for machine reassignment problem," *Comput. Oper. Res.*, vol. 68, pp. 16–29, Apr. 2016.
[18] M. Mrad, A. Gharbi, M. Haouari, and M. Kharbeche, "An optimization-based heuristic for the machine reassignment problem," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 115–132, Jul. 2016.
[19] A. Turky, N. R. Sabar, and A. Song, "Neighbourhood analysis: A case study on Google machine reassignment problem," in *Artificial Life and Computational Intelligence*, M. Wagner, X. Li, and T. Hendtlass, Eds. Cham: Springer, 2017, pp. 228–237.
[20] G. M. Portal, "An algorithmic study of the machine reassignment problem," M.S. thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2012.
[21] Y. Malitsky, D. Mehta, B. O'Sullivan, and H. Simonis, "Tuning parameters of large neighborhood search for the machine reassignment problem," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Lecture Notes in Computer Science), vol. 7874, C. P. Gomes and M. Sellmann, Eds. Berlin, Germany: Springer, 2013, pp. 176–192.
[22] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Principles and Practice of Constraint Programming—CP 2009*. Berlin, Germany: Springer, 2009, pp. 142–157.
[23] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "Isac–instance-specific algorithm configuration," in *Proc. Conf. 19th Eur. Conf. Artif. Intell.* Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010, pp. 751–756. [Online]. Available: http://dl.acm.org/citation.cfm?id=1860967.1861114
[24] H Gavranovic and M. Buljubašic, "An efficient local search with noising strategy for Google machine reassignment problem," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 19–31, 2016.
[25] F. Butelle, L. Alfandari, C. Coti, L. Finta, L. Létocart, G. Plateau, F. Roupin, A. Rozenknop, and R. W. Calvo, "Fast machine reassignment," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 133–160, 2016.
[26] F. Brandt, J. Speck, and M. Völker, "Constraint-based large neighborhood search for machine reassignment—A solution approach to the ROADEF/EURO challenge 2012," *Ann. Oper. Res.*, vol. 242, no. 1, pp. 63–91, 2016.
[27] A. Turky, N. R. Sabar, A. Sattar, and A. Song, "Parallel late acceptance hill-climbing algorithm for the Google machine reassignment problem," in *AI 2016: Advances in Artificial Intelligence*, B. H. Kang and Q. Bai, Eds. Cham: Springer, 2016, pp. 163–174.
[28] E. K. Burke and Y. Bykov, "A late acceptance strategy in hill-climbing for exam timetabling problems," in *Proc. PATAT Conf.*, Montreal, QC, Canada, 2008, pp. 1–7.
[29] A. Turky, N. R. Sabar, and A. Song, "An evolutionary simulating annealing algorithm for Google machine reassignment problem," in *Intelligent and Evolutionary Systems*. Cham, Switzerland: Springer, 2017, pp. 431–442.
[30] A. Turky, N. R. Sabar, A. Sattar, and A. Song, "Multi-neighbourhood great deluge for Google machine reassignment problem," in *Simulated Evolution and Learning* (Lecture Notes in Computer Science), vol. 10593, Y. Shi, K. C. Tan, M. Zhang, K. Tang, X. Li, Q. Zhang, Y. Tan, M. Middendorf, and Y. Jin, Eds. Cham, Switzerland: Springer, 2017, pp. 706–715.
[31] A. Turky, N. R. Sabar, S. Dunstall, and A. Song, "Hyper-heuristic based local search for combinatorial optimisation problems," in *AI 2018: Advances in Artificial Intelligence* (Lecture Notes in Computer Science), vol. 11320, T. Mitrovic, B. Xue, and X. Li, Eds. Cham, Switzerland: Springer, 2018, pp. 312–317.
[32] A. Turky, "Bi-level hyper-heuristic approaches for combinatorial optimisation problems," Ph.D. dissertation, School Sci., RMIT Univ., Melbourne VIC, Australia, 2019.
[33] A. Turky, N. R. Sabar, A. Sattar, and A. Song, "Evolutionary learning based iterated local search for Google machine reassignment problems," in *Simulated Evolution and Learning* (Lecture Notes in Computer Science), vol. 10593, Y. Shi, K. C. Tan, M. Zhang, K. Tang, X. Li, Q. Zhang, Y. Tan, M. Middendorf, and Y. Jin, Eds. Cham, Switzerland: Springer, 2017, pp. 409–421.
[34] T. Zhang, G. Su, C. Qing, X. Xu, B. Cai, and X. Xing, "Hierarchical lifelong learning by sharing representations and integrating hypothesis," *IEEE Trans. Syst., Man, Cybern. Syst.*, early access, Feb. 27, 2019, doi: 10.1109/TSMC.2018.2884996.

DARIO CANALES received the bachelor's and master's degrees in computer science from Universidad Técnica Federico Santa María (UTFSM), Valparaíso, Chile, in 2015 and 2018, respectively. After graduation, he started to work with the National Institute of Industrial Property, Chile, as a Software Architect and Developer. His current major areas of interest are related to software architectural design and artificial intelligence.

NICOLÁS ROJAS-MORALES (Member, IEEE) received the Ph.D. degree from Universidad Técnica Federico Santa María (UTFSM), Chile, in 2018. He is currently a Young Researcher in computing science with the Department of Computer Science, UTFSM. He has published technical articles in high-level heuristics search conferences such as GECCO, ANTS, and CEC. His current research interests include the foundations and applications of heuristic search methods, opposition-inspired learning strategies, parameter setting problems, and the applications to combinatorial optimization. He has also served as a reviewer of different conferences in evolutionary computation.

MARÍA-CRISTINA RIFF received the Engineering degree in computer science from Universidad Técnica Federico Santa María (UTFSM), Chile, in 1988, the Ph.D. degree in computer science and mathematics from the École Nationale des Ponts et Chaussées, in 1997, and the Habilitation to Direct Research (HDR) degree in computer science from the University of Paris-Sud XI, Orsay, France, in 2014. She was with INRIA Sophia-Antipolis, France. She supervised many master's and Ph.D. degrees students. She is currently an Adjunct Professor with the UTFSM. Her research efforts have been culminated in many refereed journals and conference publications. Her research interest includes artificial intelligence for problem solving.