Noname manuscript No.
(will be inserted by the editor)

A Comparative Study of Multi-Objective Machine Reassignment Algorithms for Data Centres

Takfarinas Saber · Xavier Gandibleux · Michael O'Neill · Liam Murphy · Anthony Ventresque

Received: date / Accepted: date

Abstract At a high level, data centres are large IT facilities hosting physical machines (servers) that often run a large number of virtual machines (VMs)—but at a lower level, data centres are an intricate collection of interconnected and virtualised computers, connected services, complex service-level agreements. While data centre managers know that reassigning VMs to the servers that would best serve them and also minimise some cost for the company can potentially save a lot of money—the search space is large and constrained, and the decision complicated as they involve different dimensions. This paper consists of a comparative study of heuristics and exact algorithms for the Multi-objective Machine Reassignment problem. Given the common intuition that the problem is too complicated for exact resolutions, all previous works have focused on various (meta)heuristics such as First-Fit, GRASP, NSGA-II or PLS. In this paper, we show that the state-of-art solution to the single objective formulation of the problem (CBLNS) and the classical multi-objective solutions fail to bridge the gap between the number, quality and variety of

T. Saber
School of Computer Science, University College Dublin, Ireland
E-mail: takfarinas.saber@ucd.ie

X. Gandibleux
IRCCyN UMR CNRS 6597, UFR Sciences, Université de Nantes, France
E-mail: xavier.gandibleux@univ-nantes.fr

M. O'Neill
NCRA, School of Business, University College Dublin, Ireland
E-mail: m.oneill@ucd.ie

L. Murphy
School of Computer Science, University College Dublin, Ireland
E-mail: liam.murphy@ucd.ie

A. Ventresque
Lero@UCD, Complex Software Lab, School of Computer Science, University College Dublin, Ireland
E-mail: anthony.ventresque@ucd.ie

solutions. Hybrid metaheuristics, on the other hand, have proven to be more effective and efficient to address the problem – but as there has never been any study of an exact resolution, it was difficult to qualify their results. In this paper, we present the most relevant techniques used to address the problem, and we compare them to an exact resolution (ϵ-Constraints). We show that the problem is indeed large and constrained (we ran our algorithm for 30 days on a powerful node of a supercomputer and did not get the final solution for most instances of our problem) but that a metaheuristic (GeNePi) obtains acceptable results: more (+188%) solutions than the exact resolution and a little more than half (52%) the hypervolume (measure of quality of the solution set).

Keywords Machine Reassignment · Metaheuristics · Multi-objective.

1 Introduction

Data centres are facilities dedicated to hosting many computer resources, and while they have been around for decades, they are now the centre of (a lot of) attention as they are increasingly the crucial element of our digital lives (e.g., the Cloud). Data centres evolve constantly as for instance machines age and are eventually decommissioned, new ones (more powerful) are bought regularly, and processes hosted are updated to potentially more greedy ones. Data centre managers adapt their systems to these evolutions and migrate processes from one machine to another one following technical and non-technical constraints and preferences. This is what we call *reassignment of processes to machines*. For instance, managers may want to increase the reliability of their data centres and move the workload from overloaded machines to less loaded and/or more powerful ones. Often, they also try to move the workload to power efficient machines, to lower the cost and environmental impact of the data centres.

One problem is that machines can range to up to tens of thousands (e.g., OVH, a European leader in the domain, have 150,000 servers in 12 data centres[1]), and services up to millions (e.g., VMware ESX accepts up to 320 VMs per host). At this scale, any instance of the reassignment problem becomes a challenge to the existing heuristics/solvers and finding the 'best' (re-) assignment an illusion. Another problem is that, as we mentioned in the previous paragraph, managers have different perspectives on what is a 'good' solution, and ranking all the solutions according to a single utility function (e.g., minimising energy consumption) is probably not relevant.

This is a perfect example of a problem where *multi-objective decision making* makes sense: an optimisation problem with various independent objectives that only decision-makers can compare – possibly collectively. For instance, Li et al. (2013) describe such an enterprise environment where managers of hosting departments have various perspectives when it comes to placement decisions. Hence we call the problem we address in this paper *Multi-Objective Optimisation for the Machine Reassignment Problem* (MOMRP). While this

[1] Source: http://www.ovh.com/fr/backstage/ – accessed on 16/05/2018.

problem has been addressed in the context of machine assignment (Mills et al., 2011), or for dynamic assignment of a small number of machines (Xu and Fortes, 2011), it has not been in itself the topic of research in the past. In this paper, we identify three objectives for the problem: (i) reliability, i.e., a penalty is given to assignments that load too much the machines; (ii) migration, i.e., assignments that move processes too much (especially to remote locations) are penalised; and (iii) electricity: trying to obtain assignments that minimises the (electrical) cost of running the data centre.

In this paper, we show that classical solutions do not perform well against this problem, in terms of the number of non-dominated solutions found (the *quantity* of solutions) or the hypervolume (Zitzler and Thiele, 1998) of the search space area defined by the Pareto frontier (the *quality* of the solutions). Algorithms of the First Fit family (e.g., First Fit decreasing, Random Fit, First Fit descent bin-balancing (Li et al., 2014)) tend to fail to satisfy the large number of constraints of the problem and have poor results as soon as the instances of the problem become large enough (and realistic). The state-of-the-art mono-objective machine reassignment algorithm, i.e., Constraint-Based Large Neighbourhood Search (CBLNS) (Mehta et al., 2011) finds solutions that improve the initial assignment, but these solutions lack diversity as they are focused around one area of the search space (low quality). Pareto Local Search (PLS) (Angel et al., 2004; Basseur, 2006; Alsheddy and Tsang, 2010) usually finds solutions but they are grouped in one area of the search space (small hypervolume), and it is 'by nature' a slow algorithm. NSGA-II (Deb et al., 2002) needs a good initial population to operate properly, while here it gets only one solution: the initial assignment. GRASP (Feo and Resende, 1995) does not perform well either in such large search spaces and ends up trying a lot of non-feasible combinations, eventually finding few or no solution at all. We describe a novel hybrid algorithm called *GeNePi* (Saber et al., 2014a), which uses successfully three steps: a first step (inspired from GRASP) to explore quickly the whole search space, a second (using NSGA-II) to introduce some variety and quality in the solutions and a last one (PLS-based) to increase the number of solutions. GeNePi outperforms all the algorithms above and some classical bin packing ones, finding nearly 5 times more non-dominated solutions on average than non-hybrid algorithms and covering the search space better with more than 100% hypervolume on average than the best non-hybrid techniques).

The comparison against other hybrid metaheuristics illustrates the importance of having a three-step method (a greedy algorithm, a genetic algorithm and a local search) with more than 2 times improvement in terms of number of non-dominated solutions and nearly 16% increase in hypervolume when compared against the second best hybrid metaheuristic.

Now, while we know that one hybrid metaheuristic outperforms the other algorithms, it is difficult to assess the efficiency and effectiveness of GeNePi in absolute terms. We have implemented an exact resolution (ϵ-Constraints method (Mavrotas, 2009)) of the problem in the same instances used for the comparison of the heuristic algorithms. We ran our implementation for up to

30 days (depending on the instance of the problem) on one node of a supercomputer. We observe that GeNePi gets more non-dominated solutions than the exact resolution (+188%) and achieves a little more than half the hypervolume of the exact resolution (52%). GeNePi also succeeds in keeping its execution time low, as it is tens of thousands of times faster than ϵ-Constraints. GeNePi is even faster or in the same order of magnitude as a single iteration of ϵ-Constraints.

In summary, in this paper we make the following contributions:

- We formally define the multi-objective machine reassignment problem.
- We show that classical multi-objective and state-of-the-art mono-objective solutions do not perform well on the multi-objective optimisation version for the machine reassignment problem.
- We describe our novel three-step algorithm called GeNePi and show that GeNePi outperforms the other algorithms in terms of number and quality of solutions.
- We implement the ϵ-Constraints exact method and run it for 30 days. We show that GeNePi achieves good results in comparison to the ϵ-Constraints method while not requiring as much execution time.

In the rest of this paper we first start by reviewing some of the work related to machine reassignment (Section 2), then we give a formal definition of the problem, with the constraints and the three objectives that we identified as the most relevant (Section 3). Next, we describe GeNePi, our algorithm for solving this MOMRP (Section 4). After this, Section 5 proposes an evaluation of the different state-of-the-art algorithms. Next, we describe the ϵ-Constraints method which provides an exact resolution of the problem, and we compare GeNePi against it (Section 7). Finally, we make some concluding remarks (Section 8).

2 Related Work

In this section, we present a survey of the literature relevant to our study: d-dimensional vector bin packing, machine reassignment and multi-objective reassignment.

2.1 d-Dimensional Vector Bin Packing

The d-dimensional vector bin packing consists of packing a set of items of various sizes, in the least number of homogeneous bins. In the case of VMs and servers, each dimension of the space represents an independent resource. Vector bin packing has been a very popular challenge in computer science and engineering, for instance in the system domain (Graham, 1972).

In the context of 2-dimensional vector bin packing, Caprara and Toth (2001) developed a set of heuristics and exact solutions, which were since

outperformed by an approximation algorithm proposed by Kellerer and Kotov (2003) with a performance ratio of 2 in the worst-case.

Beck and Siewiorek (1996) modelled the problem with more than 2 resources and made a thorough evaluation of different algorithms for assigning tasks in a multiprocessor computer – whereas Leinberger et al. (1999) focus on systems running tasks in parallel.

Jansen and Öhring (1997) studied the d-dimensional vector bin packing with conflicting items and proposed some approximation algorithms, while Gendreau et al. (2004) proposed several heuristics and lower bounds which take into consideration this set of constraints.

Most Recent works deal with the d-dimensional vector bin packing for storing multi-media content. They either study different algorithms (e.g., heuristics from the First-Fit family (Panigrahy et al., 2011)) or try to find a better approximation algorithm (Shachnai and Tamir, 2012).

2.2 Machine Reassignment

While the d-dimensional vector bin packing aims at reducing the number of bins, the machine reassignment problem (MRP) considers moving items between an already given set of bins (a.k.a., machines).

In the context of managing resources in cloud environments, several optimisation problems have been defined and studied (Mann, 2015). Doddavula et al. (2011) worked on optimising resource utilisation by reassigning tasks to different machines by comparing commonly used First Fit algorithms. Beloglazov et al. (2012) worked on the reduction of data centres' energy footprint and proposed resource allocation heuristics to tackle this problem. Stillwell et al. (2012) pushed the boundaries further by considering a resource allocation with heterogeneous machines.

Due to the increasing popularity and scale of the cloud, Google (one of the leaders on the market) proposed a challenge at the ROADEF/EURO (2012) forum with a detailed formulation for the MRP and a real-life instance scales. The challenge attracted many participants, with algorithms of different types. Although the majority were based on a local search Local Search (LS) (Gavranović et al., 2012), Large Neighbourhood Search (LNS) (Brandt et al., 2016), Variable Neighbourhood Search (VNS) (Butelle et al., 2016) or Multi-Start Local Search (MS-LS) (Masson et al., 2013), there were others based on Greedy Randomized Adaptive Search (GRASP) (Gabay and Zaourar, 2012), Simulated Annealing (SA) (Portal et al., 2012), or a combination of either Constraints Programming (CP) or Mixed-integer linear programming (MILP) solvers, with some other optimisation solutions (e.g., Local Search with either a CP (Mehta et al., 2011), or a MILP solver (Jaśkowski et al., 2015)).

The problem proposed by Google to the ROADEF/EURO challenge continued to attract researchers even years after its end to work on it and push the optimisation boundaries set by the initial state-of-the-art. Some of the works tried to improve the solutions that were submitted to the challenge

while others designed and evaluated new algorithms. Malitsky et al. (2013) enhance the work of Mehta et al. (2011) by finely tuning parameters of the large neighbourhood search to get an effective Constraint Programming Based Large Neighbourhood Search which achieves near-optimal results. Hoffmann et al. (2015) propose a hyperheuristic (i.e., a combination of several metaheuristics) that is inspired by the Simulated Annealing algorithm and that is composed of two levels of heuristics.

Recently, Turky et al. (2016) proposed an evolutionary parallel Late Acceptance Hill Climbing algorithm. The same group of authors developed a year later an Evolutionary Simulated Annealing (ESA) algorithm (Turky et al., 2017a) by replacing the Late Acceptance Hill Climbing algorithm with a Simulated Annealing. In another work from Turky et al. (2017b), the authors do not aim at finding the best solution but instead study the different neighbourhood structures that are employed in Large Neighbourhood Searches to generate and evaluate their neighbourhoods and show that they have an impact on the performance of these algorithms. In their very latest work on the topic (Turky et al., 2018), the authors combine several components in a cooperative evolutionary heterogeneous simulated annealing (CHSA) which allows them to achieve higher quality solutions.

2.3 Multi-Objective Machine Reassignment

The multi-objective consolidation (reassignment) of processes / Virtual Machines (VMs) in a data centre has been described recently as an essential research challenge for data centres (Beloglazov et al., 2012). Although we can find many works dealing with this problem in the literature (Saber, 2017; Donyagard Vahed et al., 2019), most of them either consider systems of small and unrealistic scales, or use a 'weak' multi-objective formulation (e.g., combining the objectives using a weighted-sum (Jung et al., 2010) or limiting their study to only a bi-objective resolution (Xu and Fortes, 2011)).

The MOMRP is a novel optimisation problem. It is largely inspired by the MRP proposed by Google but considers objectives that are relevant to data centres' managers in non-aggregated fashion. The first attempt at modelling it and creating an algorithm to tackle it (Saber et al., 2014a) was quite recent. Saber et al. (2015b) proposed a linear formulation for the same multi-objective problem and studied the usability of a MILP solver, but was only limited to the smallest instances. The authors also extended the work to a combination of metaheuristics and exact solutions (Saber et al., 2017). A formulation similar to the MOMRP has also been studied in the context of decentralised data centres (Saber et al., 2014b, 2015a), which considers that each site is independent when making its machine placement. The study provides a full model of the MOMRP in such infrastructures and conducts a thorough evaluation of several algorithms. The most recent formation of the problem in a decentralised infrastructure (Saber et al., 2018b) considers the possibility of decommissioning workload to public clouds with varying pricing schemes.

3 Problem Definition

The *Multi-Objective Machine Reassignment Problem* consists of optimising the usage of a set of machines $\mathcal{M}$ according to various objectives. Any reassignment has to satisfy constraints (often in large number) of the system and find a new machine $M(p)$ for every process p in the set of existing processes $\mathcal{P}$, initially placed in machine $M_0(p)$. The multi-objective reassignment tries to find non-dominated solutions (better than every other solution in some directions of the space). In some cases $M_0(p) = M(p)$, which means that the process $p \in \mathcal{P}$ does not move during the reassignment.

The model we describe below is loosely inspired by several works e.g., (Lopes et al., 2015) for an integer programming model, among which the problem definition of the ROADEF challenge (2012) has an important place.

3.1 Reassignment Problem

A machine $m \in \mathcal{M}$ belongs to a location $l \in \mathcal{L}$ (the site where the server is located). It is also in a neighbourhood $N(m) \in \mathcal{N}$ with $N(m) \subseteq \mathcal{M}$, which represents a set of machines with which it is linked to by fast connections or with which it shares the same protocol. Each machine belongs to one and only one location and one neighbourhood. Every machine m has also several resources $r \in \mathcal{R}$ (e.g., RAM, CPU, disk), in limited capacities $Q_{m,r}$. We consider that the quantity of resource r that the process p needs is fixed to $d_{p,r}$ and corresponds to a VM parameter/SLA[2]. The first constraint of the system describes the resource capacities of the machines m as limiting the resource demands of the processes p hosted on them.

$$\sum_{p \in \mathcal{P} \mid M(p)=m} d_{p,r} \leq Q_{m,r}, \quad \forall m \in \mathcal{M}, \forall r \in \mathcal{R} \tag{1}$$

The reassignment of a process is achieved using a live migration, meaning that the process is transferred to the final machine while keeping it running on its initial one. Some resources are called *transient*: $r \in \mathcal{TR} \subseteq \mathcal{R}$. Such resources (e.g., RAM and disk) are needed on both machines (initial and final machines) during a live migration, as the processes use the resources on both machines during the reassignment.

$$\sum_{p \in \mathcal{P} \mid M_0(p)=m \vee M(p)=m} d_{p,r} \leq Q_{m,r}, \quad \forall m \in \mathcal{M}, \forall r \in \mathcal{TR} \tag{2}$$

Other resources are called *non-transient*: $r \in \mathcal{NTR} = \mathcal{R} \setminus \mathcal{TR}$.

Services/applications are often multi-tier (e.g., to separate concerns) and replicated (for performance and security reasons), so it is realistic to assume

[2] a Service-Level Agreement (SLA) is a contract agreed between a data centre provider and a customer which describes the service provided (e.g., allocated resources, time to recover after an outage).

here that processes (the atomic element of workload) are organised by services. It is common for services to have an *anti-cohabitation* constraint (Bin et al., 2011), i.e., the processes composing a service cannot share the same host – for some reliability, security and performance reasons. Let $\mathcal{S}$ the set of services, then the *anti-cohabitation* constraint can be expressed as in (3).

$$\forall p_i, p_j \in \mathcal{P}, i \neq j, \forall s \in \mathcal{S}, (p_i, p_j) \in s^2 \Rightarrow M(p_i) \neq M(p_j) \tag{3}$$

For the same reasons of reliability, security and performance, services require that the number of locations hosting at least one process has to be greater than a certain number, called *spread number* and denoted σ. This allows increasing the resilience in case of failure of a data centre: the bigger the spread number σ_s, the safer the service $s \in \mathcal{S}$.

$$\sum_{l \in \mathcal{L}} \min\left(1, |\{p \mid p \in s \ \wedge \ M(p) \in l\}|\right) \geq \sigma_s, \quad \forall \ s \in \mathcal{S} \tag{4}$$

Services can also depend on each other and in this case the processes of these services need to be close to each other – to increase the performance of the system. Let $\mathcal{D}$ be the set of service dependencies in the system and we denote any services dependency with $\hookrightarrow$. Of course, as the dependencies between services can be complex, the assignment can be tricky: a process $p \in \mathcal{P}$, belonging to service $s_i \in \mathcal{S}$ which is dependent on service $s_j \in \mathcal{S}$ and service $s_k \in \mathcal{S}$, needs to be assigned to a machine $M(p') \in N(M(p))$ such that $p' \in s_j \cap s_k$.

$$\forall s_i, s_j \in \mathcal{S}, \ s_i \hookrightarrow s_j \implies \forall p_u \in s_i, \ \exists p_v \in s_j \mid N(M(p_u)) = N(M(p_v)) \tag{5}$$

Figure 1 shows graphically a scenario (i.e., instance and initial solution) of the problem. Note that resource capacities and demands are not represented here to make it simpler to understand.

Definition 1 (Machine Reassignment) An assignment A of processes to machines is a mapping: $A : \mathcal{P} \mapsto \mathcal{M}$, such that $A(p, \mathcal{M}) \rightarrow m$, which satisfies all the previous constraints 1, 2, 3, 4 and 5.
A reassignment is a function that modifies an initial assignment: $ReA : A \mapsto A$ and gives a new assignment of processes to machines.

3.2 Objectives

As said in the introduction, there are several perspectives on the best optimisation, which translate in our case into several objectives. Some studies (Purshouse and Fleming, 2007) show that a large number of objectives decreases drastically the performance of evolutionary algorithms, and that decision-makers tend to favour a small number of dimensions. We focus here on three objectives: electricity cost, VM migration cost and reliability cost, as they are recognised in the literature (Filani et al., 2008; Voorsluys et al., 2009; Schroeder and Gibson, 2010) and make sense in practice. The multi-objective

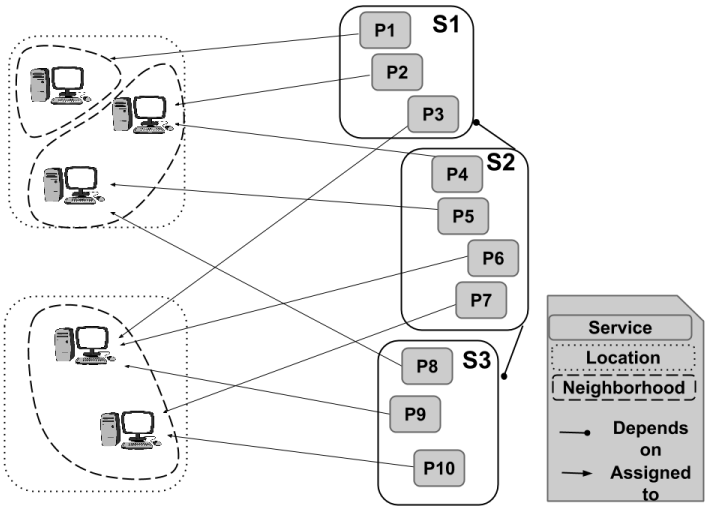

Fig. 1 Simple scenario of a correct assignment of processes to machines (spread= 2).

variant of the Machine Reassignment Problem (see Definition 1) consists of minimising the cost functions defined by the objectives.

There are many elements that can help data centre operators to predict the risk of failure of a server: to name a few the age of a machine, the vendors of its parts (e.g., processor maker) and the past history of similar machines. They are complex to collect and understand, and we do not know exactly how to process them to obtain an objective that the data centre operators and decision-makers could use (the literature seems uncertain on the matter (Schroeder and Gibson, 2010)). One thing we know is that as opposed to the risk of failure, the *reliability* is easier to compute and gathers fewer questions. Machines do operate better when they are not too loaded, and reliability can be estimated through the load: the more loaded a machine, the greater the risk of performance issues or failures.

Definition 2 (Reliability Cost) A machine $m \in \mathcal{M}$ is reliable if it is not loaded more than a reliability value $\rho(m, r)$ for each resource $r \in \mathcal{R}$, and we compute a reliability cost of m, $\rho(m)$, as:

$$\rho(m) = \sum_{r \in \mathcal{R}} \max \left(0 \ , \sum_{p \in \mathcal{P} \ | \ M_0(p)=m \ \vee \ M(p)=m} d_{p,r} \ - \ \rho(m, r) \right) \quad (6)$$

If the *safety capacity*[3] of m for the resource r is higher than the sum of the demands, then it does not impact the safety of the machine.

[3] The concept of *safety capacity* is introduced in the Google/ROADEF/EURO challenge (2012): if one or several resources of a machine are over-loaded then the machine may not be able to satisfy its SLAs.

Migrating a process has a cost which is often neglected by research in the area but is well known by practitioners (Voorsluys et al., 2009). Basically, this consists of the time needed to prepare a process p for a migration ($\mu_1(p, M_o(p))$), to transfer p ($\mu_2(p, M_o(p), M(p))$) and to install p on a new machine ($\mu_3(p, M(p))$). All these costs are dependent on some process parameters (e.g., size of the data stored on disk and RAM, complexity of the installation) and topology parameters (e.g., number of hops, bandwidth), that we do not evaluate in this paper.

Definition 3 (Migration Cost) The cost of migrating a process $p \in \mathcal{P}$ from a machine $M_0(p)$ to a machine $M(p)$ is defined:

$$\mu(p, M_o(p), M(p)) = \mu_1(p, M_o(p)) + \mu_2(p, M_o(p), M(p)) + \mu_3(p, M(p)) \quad (7)$$

Electricity cost of running machines accounts for up to 50% of their operating costs (Filani et al., 2008) and it is a burden for countries' electricity production systems: in 2007, Western European data centres consumed 56 TWh of electricity, and this is expected to double (104 TWh, or about 4 times the annual production or Ireland) by 2020 (Commission, 2007). There is a global trend towards more greener and power-aware practices, and this will certainly lead to an increase in the electricity price and other incentives for data centre managers to minimise their electricity consumption. Modelling electricity cost is complex but we follow the general assumption that states that it is a linear function of its CPU usage (Xu and Fortes, 2010; Lien et al., 2007). We then just define two parameters, α_m (linear factor) and β_m (fixed cost of running m with any given load on the CPU) for every machine m. This does not take into account other elements that may be relevant but are somehow out of the scope of our study here (e.g., cooling of data centres).

Definition 4 (Electricity Cost) The electricity cost of a machine $m \in \mathcal{M}$ in the location $l \in \mathcal{L}$ depends on the variables α_m, β_m (electricity consumption constants) and γ_l (electricity cost in l), and is expressed by the following formula:

$$\epsilon(m) = \begin{cases} \gamma_l \times \left(\alpha_m \times \sum_{p \in \mathcal{P} | M(p) = m} d_{p,CPU} \; + \; \beta_m \right) & \text{if m is running} \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

Note that a machine $m \in \mathcal{M}$ is considered running if at least one process is reassigned to it. Conversely, a machine $m \in \mathcal{M}$ is considered off if and only if no process is reassigned to it.

4 Description of our Solution: GeNePi

GeNePi applies successively three optimisation algorithms: GRASP (modified), NSGA-II and PLS. This idea of using three steps has successfully been used in different (Saber et al., 2018a; Gandibleux, 2017) areas for an approximate problem resolution, but is new in the domain of machine reassignment and not implemented in an exact way.

4.1 Ge: a variant of the constructive phase of GRASP

We use a variant of the constructive phase of *Greedy Randomized Adaptive Search Procedure* (GRASP) (Feo and Resende, 1995). Solutions are generated by trying to reassign processes one after the other, according to a greedy heuristic which is slightly relaxed to include a random factor. This method is commonly used for combinatorial problems and applied to get some quick initial solutions with good objectives. After ranking the processes according to their dependencies (if a service s_2 which contains a process p_2 depends on a service s_1 which contains a process p_1, then the process p_2 is ordered after p_1) and their resource requirements (a process with a larger resource needs is ordered before a process with a smaller resource needs), they are selected one by one. A decision of reassigning one per cent of the processes from their initial hosts has been taken, because of the tightness of transient resource constraints that limits the number of reassignments. We have noticed that without such a restriction, most of the generated reassignments were unfeasible due to the violation of transient constraints. Setting this one per cent limit reduces the number of transient constraint violations and allows the generation of more feasible solutions. While this limit seems restrictive and looks like a handicap, we saw interesting performance achievements by Ge against the regular GRASP (see Section 6). Furthermore, this limit is dropped in the following phases of GeNePi, which allows the second phase to compensate for this restriction. Note that setting this value did not undergo a full-scale parameter optimisation sweep. We believe that a better tuning of this parameter will likely yield a more significant performance improvement. The choice of the reassignment of every process is based on a linear combination of the three utility/objective functions (one per objective). Even if a linear combination of these utility functions allows us to go beyond the objective types barrier, its static definition induces getting solutions with them same objectives level of interest. This behaviour goes against the aim of a multi-objective optimisation. That is why we adopted a panel of triplet weights (λ_i, λ_j, λ_k) in $]0,1[^3$, with $\lambda_k = 1 - \lambda_i - \lambda_j$. They are chosen in such a way they cover a maximum search space by optimising the objectives separately in addition to their trade-offs. They will be used to introduce a diversification in the interest of each objective, ensuring a trade-off between them. The random part of GRASP lays in the assignment of a machine to each process, at each iteration. For each process, a set of assignable machines RM that respect the constraints is computed. A utility value U_i^m is also calculated for each machine $m \in RM$ which captures the effect of assigning the process p on m with respect to the objective $i \in \{1,2,3\}$. Then, a utility value U^m (the lower the better) is assigned to every machine $m \in RM$ using a weighted-sum: $\sum_{i=1}^{3} \lambda_i U_i^m$. We consider $minU$ and $maxU$ as respectively the lowest and largest utility obtained by any machine $m \in RM$. Therefore, we define a set of interesting reassignable machines IRM as the machines that belong to RM and have a utility lower than or equal to $(minU + (1-r) * [maxU - minU])$, with $r \in [0,1]$. A machine is randomly selected from this set of interesting machines

IRM to assign the process to it. During the assignment, it may happen that a process has no machine able to host it. The solution is declared infeasible, and removed from the initial solutions. Globally, at the end of this step, we expect to have a set of decent solutions spread over the search space.

4.2 Ne: NSGA-II

We use for this step a genetic algorithm called *Non-dominated Sorting Genetic Algorithm-II* (NSGA-II) (Deb et al., 2002). This step is useful for the improvement of the *Pareto set*[4] obtained from the first step. This metaheuristic allows to get a good dissemination of the solutions around the Pareto frontier and prevent their accumulation in some area of the search space. Hopefully, it allows GeNePi getting a smooth frontier and increases the number and the quality of the non-dominated solutions. It is a genetic algorithm, i.e., it runs an evolutionary process which matches individuals (i.e., solutions or assignments) at each generation and mixes their features (as the biological evolution would do with genes). The two main actions are *crossover* which mixes genes from two parents, and *mutation* that randomly creates individuals with new features. There exist several ways of doing crossovers, which is more or less a cut and paste operation where assignments in the set of actual solutions are split into regular length segments and swapped with one another (Falkenauer, 1998). In our case, crossovers consider the exchange of services (i.e., exchanging the assignments for all the processes belonging to a same service s between the two given solutions a_i and a_j by swapping the assignments of every process p_i in s from the solution a_i with the process p_j in s from the solution a_j) rather than blocks of process assignments—which minimises the number of crossovers that generate infeasible solutions. Of course the diversity is less than with crossovers on processes, but we compensate with a bigger probability of mutations (i.e., random assignments in solutions to see whether this improves the utilities). After a generation has "passed", some new individuals are kept (usually the fittest, those with the best objective values: low domination rank, but also some other that allow introducing some variety: high crowding-measure (Deb et al., 2002)), and others are suppressed. Hence the global population of assignments only improves (descendants worse than their parents are likely to be suppressed). Besides, last generations tend to be well distributed over the Pareto frontier.

4.3 Pi: a Pareto Local Search

Finally, we try to improve the Pareto set by using a *Pareto Local Search* (PLS) (Alsheddy and Tsang, 2010). It consists of applying several local search operators on the solutions belonging to the *Pareto* frontier. Few simple moves

[4] Pareto set: a set of non-dominated solutions (i.e., better than all other solutions in one or more objectives).

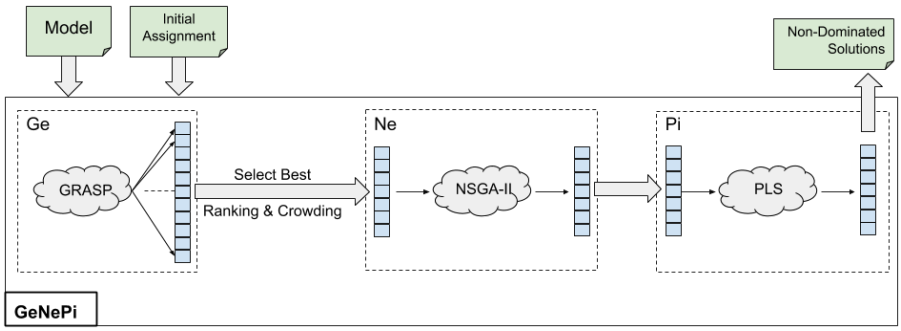

Fig. 2 Overview of the different components of GeNePi.

are chosen from the work of Gavranović et al. (2012) to analyse the neighbourhood of actual solutions: (i) swap, i.e., taking two processes and exchanging their assignments; (ii) 1-exchange, where one process at a time is selected and reassigned to any machine that accepts it; (iii) shift, where processes belonging to the same service are reassigned to the machine of their following process in a chain rotation fashion (which maintains the satisfaction on the dependency constraints). These moves allow probing of a large neighbourhood around the current solutions, which may generate some redundancies if the solutions are close of one another. To overcome this problem, we generate boxes by clustering solutions and apply a local search to the most isolated solution in each of them (i.e., has the largest crowding-measure value). Only one neighbourhood is generated for every selected solution at every iteration, even if new interesting solutions have been found. This balances the improvement and reduces the execution time as redundancy is less likely.

Figure 2 is a flowchart which shows the composition of GeNePi as three successive steps. GeNePi receives a model of the MOMRP instance in addition to the initial assignment. GeNePi uses Ge (a modified version of GRASP) as an initial step to generate multiple solutions. Then, the best solutions among them (based on the ranking and crowding metrics) are sent to Ne (an implementation of NSGA-II) to perform an evolutionary process and get better solutions. The best solutions that result of Ne are fed to the third and last step (i.e., Pe) which performs a PLS on some of the Pareto solutions. All the non-dominated solutions that are found after Pe are returned by GeNePi to the decision-makers as the final set of non-dominated solutions.

5 Experimental Setups

In this section, we evaluate the performance of our solution against other state-of-the-art multi-objective reassignment solutions, using several metrics: time, quantity (number of solutions) and quality of solutions (hypervolume). We create a benchmark[5] inspired by the ROADEF Challenge (2012).

[5] Available at: http://galapagos.ucd.ie/wiki/OpenAccess/Saber2019DatasetMOMRP

5.1 Experimental Setups

The ROADEF challenge (2012) is particularly suited to our needs, as it is rather realistic (proposed by Google) and it is quite comprehensive: a lot of resources for the machines/processes while many papers in the area only consider two (namely, RAM and CPU), reasonably high number of machines and processes, complex dependencies and constraints on the services and processes which make the assignments not straightforward. The ROADEF dataset distinguishes three categories of instances (a_1 are considered 'easy', a_2 'medium' and b 'hard').

In this paper, we pick up 14 instances (see Table 1), leaving out only the biggest ones (as they require a large execution time). We have added variables α_m and β_m to each machine $m \in \mathcal{M}$, and γ_l for every location $l \in \mathcal{L}$ in order to include electricity consumption. All our algorithms have been developed in C++. Experiments were run on a computing cluster with 24 cores 2.0GHz Intel Ivy Bridge CPU and 128GB of RAM. Furthermore, experiments with algorithms having random parameters (all algorithms except the exact ϵ-Constraints method) were repeated 10 times. Note that all our algorithms are fully sequential and do not take advantage of this parallelism with the exception of the ϵ-Constraints method which uses a mono-objective solver (i.e., CPLEX) that has enabled multi-processing capabilities.

Instance	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{TR}\rvert$	$\lvert\mathcal{M}\rvert$	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{D}\rvert$
a_1_1	2	0	4	4	79	100	1	0
a_1_2	4	1	100	4	980	1,000	2	40
a_1_3	3	1	100	25	216	1,000	5	342
a_1_4	3	1	50	50	142	1,000	50	297
a_1_5	4	1	12	4	981	1,000	2	32
a_2_1	3	0	100	1	1,000	1,000	1	0
a_2_2	12	4	100	25	170	1,000	5	0
a_2_3	12	4	100	25	129	1,000	5	577
a_2_4	12	0	50	25	180	1,000	5	397
a_2_5	12	0	50	25	153	1,000	5	506
b_1	12	4	100	10	2,512	5,000	5	4,412
b_2	12	0	100	10	2,462	5,000	5	3,617
b_3	6	2	100	25	15,025	20,000	5	16,560
b_4	6	0	500	50	1,732	20,000	5	40,485

Table 1 The dataset used for our evaluation (ID and size of the different instances). $|\mathcal{R}|$: the number of resources, $|\mathcal{TR}|$: the number of transient resources, $|\mathcal{M}|$: the number of machines, $|\mathcal{L}|$: the number of locations, $|\mathcal{S}|$: the number of services, $|\mathcal{P}|$: the number of processes, $|\mathcal{N}|$: the number of neighbourhoods, and $|\mathcal{D}|$: the number of dependencies.

5.2 Metrics

Comparing multi-objective optimisation approaches is complex as the set of solutions they give on a problem can be seen from different perspectives: cov-

erage, closeness to the Pareto frontier, variety, and many more (Zitzler et al., 2002). The problem probably comes from the fact that the Pareto frontier is unknown most of the time, and that the different objectives cannot be taken in isolation to give the quality of any solution. In this paper, we decided to take only a few unary operators as metrics (for a more comprehensive study of the various possible operators, see (Zitzler et al., 2003)): unary as they take a set of solutions and give a single value, allowing to compare the different approaches.

The first metric we use is the number of *non-dominated (efficient) solutions* and we refer to it as the quantity of solutions in the non-dominated set. Finding a large number of solutions is always better as it provides more alternatives to the decision-makers.

The other metric is the *hypervolume* (also known as the $\mathcal{S}$ metric) and was introduced by Zitzler and Thiele (1998). We sometimes call it quality of the solutions. This is a widely used metric in the area of optimisation to evaluate the performance of multi-objective algorithms that aims at understanding how the output sets are spread in the different dimensions. In short, the hypervolume measures the space (in the n dimensions of the n objectives) defined by the set of non-dominated solutions and a reference point, picked in the space as far as possible from the Pareto frontier. The bigger the hypervolume, the more interesting are the solutions in the found non-dominated solutions set, as they increase the dominated area. Fleischer (Fleischer, 2003) proved that the maximisation of the hypervolume is equivalent to finding the optimal Pareto frontier. Note that in order to compare the result sets of different algorithms, we use the same reference points for each instance of the MOMRP.

5.3 Algorithms

In our study, we compare four different types of algorithms against the baseline results when only considering the initial assignment (called Initial), running for the same period of time. The first algorithms are from the *First Fit family*. These heuristics are designed for Vector Bin Packing (Panigrahy et al., 2011) and they are considered efficient. Each of them uses an ordered sequence (by resource demands) of processes they aim to place on machines as input. We chose among them First Fit (FF) which selects the first machine that fits for every process; Random Fit (RF) which selects randomly a machine among those which fit; and First Fit Descent Bin-Balancing (BB) which selects the least loaded machine for each process.

The second set of algorithms is the state-of-the-art solutions from the multi-objective optimisation field. The first of them is GRASP in its original definition, i.e., the choice of reassigning processes to machines is based on a uniform probability distribution of the possible machines. We also evaluate the first step of GeNePi (Ge) as it is a variation of GRASP that we expect to be better than GRASP for our scenario. The last algorithm in this family is a Pareto Local Search (PLS), with a number of boxes at every iteration equals

the number of solutions in the non-dominated solutions set. Notice that we do not compare to a genetic algorithm (alone) here as we observe that running one (e.g., NSGA-II) with a random generation of its initial population could not make any improvement over the initial solution. The seed, i.e., the initial population's individuals are important for genetic algorithms.

The third set consists of the state-of-the-art mono-objective machine reassignment algorithm that was designed for the Google/ROADEF/EURO 2012 challenge, i.e., Constraint-Based Large Neighbourhood Search (CBLNS) (Mehta et al., 2011). More precisely, we use the finely tuned version of the algorithm (Malitsky et al., 2013) which achieves near-optimal results. As its name indicates, CBLNS is a hybrid algorithm which uses a combination of Large Neighbourhood Search metaheuristic (LNS) and Constraint Programming (CP). CBLNS is not multi-objective (which is also the case for other algorithms used for the ROADEF challenge) and it cannot be applied directly to our multi-objective problem. To cope with this, we adapted CBLNS and came up with a 'weak' multi-objective version using a weight-sum with a vector of equal weights.

We also evaluate different hybrid metaheuristics: (i) GrNe where we reserve a third of the execution time to GRASP in order to create an initial population and run NSGA-II in the two remaining thirds of the execution time, (ii) GeNe with an initial population obtained with our adapted greedy algorithms (i.e., Ge), and (iii) GeNePi with its three successive steps.

5.4 Statistical Analysis and Tests

To validate the significance of our comparison, we perform the non-parametric two-tailed Mann-Whitney U test (MWU). For two distinct algorithms, MWU takes in the different performance values obtained on a given metric from each run (in our case 10). MWU returns the p-value that the algorithms obtain different values. We consider tests to be significant when the p-values are below the 0.05 significance level. Furthermore, given the small number of runs in our experiment (due to the long time each of them takes), and in order to reduce the chances of having incorrect rejection of the true null hypothesis, we use a conservative but safe adjustment (i.e., the standard Bonferroni adjustment (Arcuri and Briand, 2011)) which lowers the risk of their erroneous rejection. Moreover, following the advice in the practical guide proposed by Arcuri and Briand (2011), we measure the effect of size using the the non-parametric $\hat{A}_{12}$ (Vargha and Delaney, 2000) which evaluates the ratio of runs from the first algorithm that outperform the second one. In the literature, it is considered that when $\hat{A}_{12}$ is above 0.71 that differences between the algorithms are large. Given that these significance tests can only be performed on two algorithms at a time, and to avoid the combinatorial explosion when reporting the results in our manuscript, we only report the results for GeNePi against the other best algorithm.

5.5 Tuning the steps of GeNePi

Each of the three steps composing GeNePi has several parameters that need to be tuned, and globally we need to decide how many iterations or how much time we allocate to each of them to make the best use of each. Note that our tuning has been done on one instance (a_1_5), as tuning is computationally expensive and we think the conclusions can be extended to the process in general.

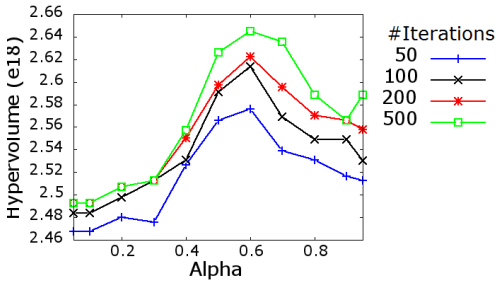

Fig. 3 Average hypervolume obtained with Ge over 10 runs on instance a_1_5 using different values for the parameter α.

The first step of GeNePi is Ge (based on GRASP), which has only one value to tune: α, the factor leading to more randomised greedy search (bigger α) or local search (smaller α). We conducted a thorough evaluation of the impact of different values of α from 0.05 to 0.95 (repeated 10 times for each value).

Figure 3 shows the average hypervolume obtained using Ge over 10 runs on instance a_1_5 when setting the parameter α to different values ranging from 0.05 to 0.95. We see that low values of α lead to a bad hypervolume and that the hypervolume increases until $\alpha = 0.6$ before decreasing slightly. Therefore, the best value of α seems to be 0.6 regardless of the number of iterations.

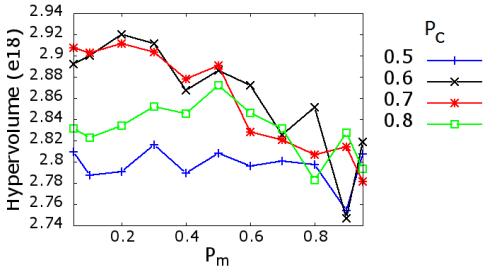

Fig. 4 Average hypervolume obtained with GeNe over 10 runs on instance a_1_5 by setting α to 0.6, size of the population to 50 and the number of iterations to 100, while varying both P_c and P_m.

For Ne (i.e., NSGA-II), we combined 9 possible values $\{0.1, 0.2, \ldots, 0.9\}$ for P_c and P_m, obtaining 81 different variations of the parameters (we again run 10 times each combination).

Figure 4 shows the average hypervolume obtained using GeNe over 10 runs on instance a_1_5 when setting α to 0.6, size of the population to 50 and the number of iterations to 100, while varying both P_c and P_m within the interval [0.05, 0.95]. Since results with $P_c < 0.5$ and $P_c > 0.8$ are not good and for readability, we only show the evolution of the hypervolume for $0.5 \leq P_c \leq 0.8$.

We realise that P_c (the probability of crossover) values between 0.6 and 0.7 give better results, while the impact of P_m (the probability of mutation) seems less important between 0.1 and 0.3. However, 0.2 gives slightly better results. We then decided to use $P_c = 0.6$ and $P_m = 0.2$.

Pi has only one parameter that we can tune here: the number of zones (boxes) that it can explore. This number of zones has an impact on the quality of the Pareto frontier, and hence on the hypervolume. A small number of zones means less neighbourhood probing, but also less redundancy and execution time, while more zones allow analysing more neighbourhoods (and to find more solutions) but there is a cost in redundancy and execution time. After performing a limited parameter sweep, we decided to use 10 zones to search for potential local non-dominated solutions as this value seems to be a good trade-off between finding more non-dominated solutions and keeping the execution time low.

Table 2 summarises the tuning parameters for each step of GeNePi that we define to provide decision-makers with a set of good solutions, covering the solutions space, in a reasonable time.

Ge (1^{st} step - GRASP)		**Ne (2^{nd} step - NSGA-II)**		**Pi (3^{rd} step - PLS)**	
α	0.6	Probability of crossover	0.6	# zones (# boxes)	10
$\|\Lambda\|$	4	Probability of mutation	0.2	# iterations	1
# iterations	100	Size of population	50		
		# iterations	100		

Table 2 Parameters for the different steps of GeNePi after a tuning study.

6 Evaluation of GeNePi against other heuristics

In this section we compare our solution, GeNePi, against other heuristics (see Section 5.3) in terms of quality (hypervolume) and quantity of solutions.

Table 3 summarises our evaluation, for all instances, all algorithms and both metrics. We also put in bold the best value for each instance and each metric. Note that all comparisons between GeNePi and the other best algorithm on every instance and metric have an adjusted Bonferroni MWU value less than 5% and an $\hat{A}_{12}$ score of 1. At first glance, we see that GeNePi outperforms other algorithms in both number of solutions and hypervolume.

First Fit family algorithms (i.e., RF, FF and BB) only work on less constrained problems as they tend to reassign many processes to machines different from their initial ones—which is likely to generate constraint violations.

The same behaviour is observed for GRASP, which tends to reassign several processes. GRASP generates a lot of solutions, but most of them end up being infeasible and violating one or several constraints. Ge, the first step of GeNePi gets a good hypervolume but not an outstanding number of solutions. This was expected as it is only an improvement of GRASP which itself suffers from a lack of solutions. Results for PLS are contrasted as they can be good in terms of quantity (better than Ge at times) but are poor in terms of quality.

GrNe, being dependent on the quality of the initial population, performs badly as genetic algorithms require a good initial population to perform well. GeNe takes advantage of the improvement made to GRASP by Ge to find 'good' and diverse solutions as an initial population. CBLNS finds new non-dominated reassignment on all instances. However, their overall hypervolume is at the same level as Ge on average. GeNePi is by far the best algorithm, and we explain it by the composition of its elements: Ge (i.e., modified GRASP) finds a large number of solutions in multiple directions of the search space, allowing Ne (i.e., NSGA-II) to operate properly and to find new solutions that balance all the objectives, while PLS, the last step, increases the number of solutions around the previously found ones.

Instance	Metric	Initial	RF	FF	BB	GRASP	Ge	PLS	GrNe	GeNe	CBLNS	GeNePi
a_1_1	#sol	1	4	10	87	20	42	10	14	106	56	**224**
	hyp e15	3.34	2.60	2.95	3.73	2.73	3.60	2.40	2.80	3.91	3.02	**3.98**
a_1_2	#sol	1	–	–	–	–	26	2	–	44	36	**182**
	hyp e16	7.49					8.47	7.49		8.92	8.25	**9.22**
a_1_3	#sol	1	–	–	–	–	19	2	–	27	29	**132**
	hyp e16	4.17					4.27	4.17		4.30	4.25	**4.32**
a_1_4	#sol	1	–	–	–	–	40	2	–	75	58	**136**
	hyp e16	9.72					11.10	9.72		12.10	11.00	**12.17**
a_1_5	#sol	1	4	10	2	14	49	32	16	112	15	**282**
	hyp e18	2.42	2.51	2.51	2.45	2.59	2.74	2.52	2.57	2.92	2.69	**3.15**
a_2_1	#sol	1	33	41	–	69	57	4	71	152	35	**231**
	hyp e19	4.57	4.86	4.95		5.41	5.43	4.61	5.46	5.83	5.06	**5.93**
a_2_2	#sol	1	–	–	–	–	22	2	–	41	8	**197**
	hyp e20	1.33					1.55	1.33		1.68	1.51	**1.72**
a_2_3	#sol	1	–	–	–	–	30	67	–	57	21	**202**
	hyp e18	2.02					2.36	2.04		2.59	2.31	**2.66**
a_2_4	#sol	1	–	–	–	–	28	2	–	80	26	**253**
	hyp e18	6.42					7.62	6.42		8.68	7.09	**9.07**
a_2_5	#sol	1	–	–	–	–	28	2	–	76	14	**220**
	hyp e18	9.91					10.30	9.91		10.80	10.29	**10.90**
b_1	#sol	1	–	–	–	–	27	39	–	58	43	**242**
	hyp e20	8.20					8.34	8.34		8.50	8.38	**8.53**
b_2	#sol	1	–	–	–	–	23	2	–	93	7	**300**
	hype 21	1.43					1.48	1.43		1.51	1.46	**1.53**
b_3	#sol	1	–	–	–	–	20	108	–	43	23	**162**
	hyp e21	6.25					6.27	6.27		6.298	6.28	**6.301**
b_4	#sol	1	–	–	–	–	22	3	–	31	12	**118**
	hyp e21	3.65					3.67	3.67		3.69	3.67	**3.70**

Table 3 Summary of average over 10 runs of number of solutions found (#sol) and hypervolume.10^x (hyp) for the various algorithms and each ROADEF instance we use. For both metrics, the higher the better. We put in bold the best values for each instance. Note that all the comparisons between GeNePi and the other best algorithm have an adjusted Bonferroni MWU value less than 5% and an $\hat{A}_{12}$ score of 1. ('–' indicates that the algorithm does not improve on the initial assignment)

Table 4 summarises the improvement of GeNePi in comparison to the second best algorithm on both hypervolume and number of non-dominated solutions. We show these results with the hybrid methods (comparison done against all algorithms including those combining more than one technique, i.e., GrNe, GeNe and CBLNS) and without the hybrid methods (comparison done only against algorithms composed of one technique, i.e., RF, FF, BB, GRASP, GE and PLS).

Table 4 shows that GeNePi significantly outperforms all non-hybrid algorithms with an increase of nearly 400% in non-dominated solutions and of more than 100% in hypervolume on average. It also shows that GeNePi achieves at least an improvement of 50% in terms of number of non-dominated solutions and at least more than 49% in terms of hypervolume against these same non-hybrid algorithms.

Table 4 shows that GeNePi also outperforms hybrid metaheuristics on every single instance, both quantitatively (more than 108% non-dominated solutions on average) and qualitatively (more than 15% increase in hypervolume on average).

	% Improvement of GeNePi against 2^{nd} best algorithm			
	# Solutions		**Hypervolume**	
	w/o hybrid	**w hybrid**	**w/o hybrid**	**w hybrid**
a_1_1	157.47	111.32	63.15	11.90
a_1_2	600.00	313.64	76.91	21.05
a_1_3	594.74	355.17	49.46	15.26
a_1_4	240.00	81.33	77.80	2.95
a_1_5	475.51	151.79	129.22	46.25
a_2_1	234.78	51.97	57.91	8.28
a_2_2	795.45	380.49	78.39	11.53
a_2_3	201.49	201.49	87.04	12.18
a_2_4	803.57	216.25	120.90	17.26
a_2_5	685.71	189.47	155.54	11.29
b_1	520.51	317.24	139.64	10.13
b_2	1,204.35	222.58	91.28	23.59
b_3	50.00	50.00	139.64	7.00
b_4	436.36	280.65	150.00	25.00
Average	**400.00**	**108.81**	**101.18**	**15.98**

Table 4 Summary of the improvement (in per cent) obtained using GeNePi on both number of non-dominated solutions and hypervolume when applied on the different ROADEF instances. The table includes results with (w) and without (w/o) taking into account hybrid algorithms.

One of the challenges here is that the execution time is limited: even if the reassignment is done on a monthly or a quarterly basis, as it often happens, the decision process is complex and decision-makers cannot wait more than a few hours or days: they verify and modify the solutions to suit their needs before making any decision.

Table 5 shows the average execution time over 10 runs of the studied algorithms on the different instances to obtain the aforementioned results (i.e.,

Instance	**a_1_1**	**a_1_2**	**a_1_3**	**a_1_4**	**a_1_5**
Time (s)	2	3,106	441	309	332
Instance	**a_2_1**	**a_2_2**	**a_2_3**	**a_2_4**	**a_2_5**
Time (s)	3,905	600	695	342	347
Instance	**b_1**	**b_2**	**b_3**	**b_4**	
Time (s)	14,991	10,028	39,596	63,535	

Table 5 Average execution time (s) of GeNePi over 10 runs and other evaluated algorithms on the different instances.

shown in Table 3). We notice that GeNePi works in a short time for the easy and medium instances, and in a reasonable time for the bigger ones. The 17 hours of running GeNePi for the biggest instance we consider (b_4) are totally justified if this can save money, increase the reliability and do not put the data centre at risk by performing too many migrations. Especially as GeNePi can give 118 solutions for this instance, i.e., 118 options for the operators to make the most informed decision.

To give the reader a sense of what happens during the optimisation of the different algorithms, we plot the hypervolume improvement curve for the different instances and the different algorithms. Each point corresponds to one or several new non-dominated solutions found (with the timestamp of this new solution in the x-axis and the new hypervolume of the solution set in y-axis). We especially want to see here the relative impacts of the 3 phases of GeNePi.

Figure 5 shows the average hypervolume improvement curves of the different algorithms on the different instances.

We see from Figure 5 that algorithms from the First Fit family are only making some improvement for a limited number of instances.

GRASP has a somewhat similar behaviour, but achieves a better improvement when it can make any. PLS finds many solutions, but they are somehow local and thus have a marginal impact on the hypervolume in most instances. Ge brings a very good improvement at the beginning (in the first third of the execution time), but finds fewer solutions for the last 2 thirds of its execution time.

GrNe is penalised by the poor initial population obtained from the GRASP step, thus taking a large time to reach a significant hypervolume. CBLNS shows a slow but steady improvement of the hypervolume in the first half of the optimisation, but we notice a quasi-stagnation after that due to the optimisation in one direction of the search space. Algorithms which have the component Ge (i.e., GeNe and GeNePi) show a good improvement in hypervolume at the beginning. However, they cope with the lack of information between the different steps, by substituting Ge with a genetic algorithm (i.e., NSGA-II), which shows huge improvements in hypervolume in a short amount of time. However, in the same way as Ge, GeNe also plateau in most instances, which makes sense to have a wisely used local search as a third component (such as in GeNePi).

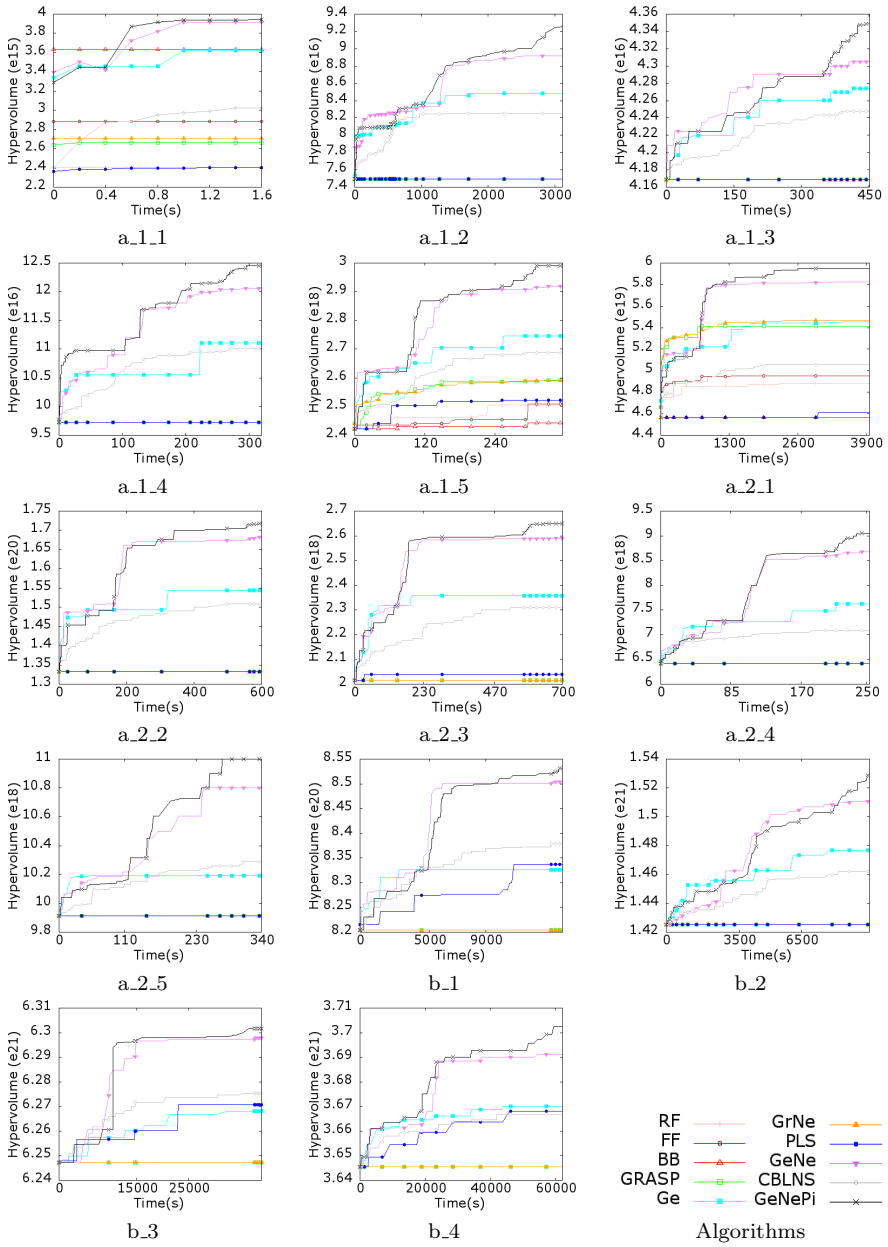

Fig. 5 Average hypervolume improvement curves of the different algorithms over 10 runs on the different instances.

7 Evaluation of GeNePi against an Exact Resolution

In this section we compare the performance of GeNePi against an exact resolution algorithm.

The aim of the multi-objective resolution is to find optimal non-dominated solutions (Pareto optimal), which are solutions that cannot be strictly dominated by others. There exist three different definitions of Pareto optimality (i.e., Pareto optimal, Weakly Pareto Optimal, and Properly Pareto Optimal). These three types of Pareto optimality have been defined by Marler and Arora (2004).

Based on the aforementioned definitions, the optimal non-dominated set can be either (i) maximal: all the respective solutions of every image (codomain, or the objective values for the given placement) in the objective research space, (ii) minimal: one solution for each image, or (iii) supported: only the solutions that have their image in the convex hull of the objective search space.

Visée et al. (1998) showed on a knapsack problem that the number of supported solutions grows linearly with the size of the problem, whereas the maximal non-dominate set grows exponentially. Furthermore, finding multiple solutions with the same objectives values might be interesting from an engineering point of view, but not ideal for comparison purposes (the quality of the Pareto front is not improved when finding multiple non-dominated solutions with the same objective values). Therefore, we chose to use the most common definition of Pareto optimality (i.e., Pareto optimal) and seek for the Minimal Optimal Pareto Front.

Several methods exist in the literature to find the minimal optimal Pareto front. In this paper we chose to compare our algorithm against the ϵ-Constraints method (Mavrotas, 2009). In addition to finding the entire Pareto front (not only solutions in the convex hull), the ϵ-Constraints method does not need to aggregate the objectives. It keeps the objectives of different types and scales independent throughout the resolution.

7.1 Description of the ϵ-Constraints Method

ϵ-Constraints method is based on the transformation of multi-objective problem into several mono-objective ones, by considering only one of the objectives and transforming the others as constraints bounded by a vector of values $\mathcal{E}$.

Let assume the following multi-objective problem:

$$\begin{aligned} \min \quad & (f_1(x), f_2(x), f_3(x)) \\ \text{s.t.} \quad & x \in \mathcal{X}. \end{aligned} \tag{9}$$

where f_i, $i \in \{1, 2, 3\}$, are three objective functions to be minimised, and $\mathcal{X}$ is the set of feasible solutions represented in a form of a vector of decisions x.

After converting the model (9) into the following model:

$$\begin{aligned} \min \quad & f_1(x) \\ \text{s.t.} \quad & f_2(x) \leq \epsilon_2 \\ & f_3(x) \leq \epsilon_3 \\ & x \in \mathcal{X}. \end{aligned} \tag{10}$$

ϵ-Constraints method solves iteratively different instances of (9) using a succession of $\mathcal{E} = \{\epsilon_2, \epsilon_3\}$ vectors, which probe the entire Pareto optimal front.

7.2 Implementation of our Problem with ϵ-Constraints

We have implemented the ϵ-Constraints method on our problem considering the objectives mentioned in Section 3 (i.e., Reliability Cost, Migration Cost and Electricity Cost). The nature of the reliability and the migration costs (i.e., integer values), makes it easier to set an adequate ϵ for each of them (setting it to the last value of the objective minus one). However the electricity cost is a real value, which makes finding a suitable ϵ impossible without taking a risk of either not finding all the non-dominated solutions (if ϵ is set to be too large) or having computational rounding errors (if ϵ is set to be too small). That is why we choose to always keep the electricity cost as the main optimised objective, while generating constraints from the other objectives with variable ϵ values.

7.3 CPLEX Solver

In our implementation of the ϵ-Constraints method, we exploit the linear aspect of our problem while solving iterative mono-objective problems. We use one of the best MILP solvers on the market: *IBM ILOG CPLEX*. In addition to its performance in comparison to other MILP solvers, CPLEX has the advantage of solving problems in a parallel fashion. Thus, fully exploiting the execution environment (i.e., one node of a computing cluster with 24 cores 2.0GHz Intel Ivy Bridge CPU and 128GB of RAM).

7.4 Results Obtained Using ϵ-Constraints

We run ϵ-Constraints on all instances used in the previous section, for a maximum of 30 days, and we extracted performance counters every 10 days. We notice from Table 6 that we only get optimal/complete Pareto front for four instances out of 14 during the 30 days (i.e., a_1_1, a_1_5, a_2_1, b_1).

Table 6 shows the results in terms of hypervolume and number of non-dominated solutions obtained when running the ϵ-Constraints method on the different instances. The hypervolume is measured every 10 days for a duration of 30 days. We see that results obtained after 10 days of execution time for both hypervolume and number of non-dominated solutions are very high in comparison to the initial assignment/placement. However, this improvement

depends on the instance we are trying to solve as the required execution time for solving the iterative mono-objective problems differs from an instance to another. The difference in execution time per mono-objective problem can be noticed based on the number of new solutions found by ϵ-Constraints on the different instances at a given timestamp. We also notice that both the quantity of solutions and their respective quality keep increasing with the execution time, which indicates that ϵ-Constraints finds more solutions on the Pareto front. However, we see that this increase in hypervolume is not linear. This is due to the fact that the iterative mono-objective problems take more time to be solved optimally using CPLEX. It is also due to the fact that they are located close to each other, making the increase in hypervolume marginal.

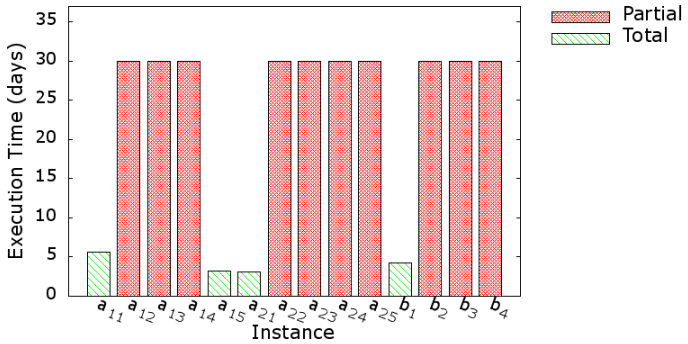

Fig. 6 Execution time in number of days of the ϵ-Constraints method on the ROADEF instances.

Table 7 shows a comparison of GeNePi and ϵ-Constraints: the numbers say how much (percentage) of the optimal solution found (within the 30 days) GeNePi performs. A number lower than 100 means that GeNePi does not reach the value of the exact resolution, while a number bigger than 100 means GeNePi outperforms the exact resolution. Table 7 shows that GeNePi outperforms ϵ-Constraints in terms of number of non-dominated solutions with respectively more than 364% (10 days), 241% (20 days) and 188% (30 days). However if we take a look at the different instances in more details, we observe that GeNePi does not always outperform ϵ-Constraints (e.g., GeNePi does poorly on a_2_2 and a_2_3). It also shows that GeNePi gets a good hypervolume compared to ϵ-Constraints reaching more than 70% of ϵ-Constraints' hypervolume after 10 days. Moreover, GeNePi's good performance w.r.t. ϵ-Constraints does not decrease over time: GeNePi still gets 54.85% after 20 days and 51.54% after 30 days of running ϵ-Constraints.

Table 8 summarises the execution time of both GeNePi and ϵ-Constraints. It also includes the average execution time per solution found during the ϵ-Constraints method, when run for 10, 20, and 30 days. We see that GeNePi is 1,000 times faster on average than ϵ-Constraints, without taking into account

		Initial	GeNePi	ϵ-Constraints		
				10 days	20 days	30 days
a_1_1	**#sol**	1	224	1280	–	–
	hyp (e15)	3.34	3.98	9.55		
a_1_2	**#sol**	1	182	37	54	78
	hyp (e16)	7.49	9.22	9.99	10.06	10.12
a_1_3	**#sol**	1	132	83	152	212
	hyp (e16)	4.17	4.32	4.75	4.79	4.8
a_1_4	**#sol**	1	136	80	149	223
	hyp (e16)	9.72	12.17	12.59	12.79	12.89
a_1_5	**#sol**	1	282	56	–	–
	hyp (e18)	2.42	3.15	4.12		
a_2_1	**#sol**	1	231	109	–	–
	hyp (e19)	4.57	5.93	6.25		
a_2_2	**#sol**	1	197	2,994	4,005	6,603
	hyp (e20)	1.33	1.72	2.77	2.81	2.88
a_2_3	**#sol**	1	202	2,890	3,421	4,173
	hyp (e18)	2.02	2.66	2.93	2.94	2.95
a_2_4	**#sol**	1	253	615	1,034	1,440
	hyp (e18)	6.42	9.07	11.43	11.82	12.02
a_2_5	**#sol**	1	220	1,355	2,478	3,516
	hyp (e18)	9.91	10.9	12.68	13.02	13.24
b_1	**#sol**	1	242	272	–	–
	hyp (e20)	8.20	8.53	10.15		
b_2	**#sol**	1	300	31	61	89
	hyp (e21)	1.43	1.53	1.52	1.52	1.53
b_3	**#sol**	1	162	246	478	714
	hyp (e21)	6.25	6.3	7.04	7.05	7.06
b_4	**#sol**	1	118	5	8	11
	hyp (e21)	3.65	3.7	3.66	3.69	3.69

Table 6 Number of non-dominated solutions (#sol) and hypervolume.10^x (hyp) for GeNePi and ϵ-Constraints on the different instances. (– indicates that the execution of ϵ-Constraints finished)

instance a_1_1, and almost 23,000 times faster on average when considering all the instances. It also shows that despite having a big variation in the execution time ratio, GeNePi is always faster than ϵ-Constraints with at least one order of magnitude. It is even faster than a single solution found by ϵ-Constraints in 9 instances out of 14, and always within the same order of magnitude in the rest of the instances. This shows that GeNePi is getting not only good results, but also with an execution time that is either faster than or in the same order of magnitude as solving one mono-objective problem (i.e., only one solution of ϵ-Constraints) with one of the best commercial MILP solvers (i.e., CPLEX) while running on a cluster node. We also notice from Table 8 that ϵ-Constraints average execution time per solution increases over time. This consolidates the aforementioned result that the iterative mono-objective problems get harder to solve over time.

	Performance of GeNePi: % of the results of ϵ-Constraints					
	# Solutions			Hypervolume		
	10 days	**20 days**	**30 days**	**10 days**	**20 days**	**30 days**
a_1_1	17.50	17.50	17.50	22.49	22.49	22.49
a_1_2	491.89	337.04	233.33	69.10	67.42	65.67
a_1_3	159.04	86.84	62.26	26.00	24.31	23.76
a_1_4	170.00	91.28	60.99	85.48	79.90	77.19
a_1_5	503.57	503.57	503.57	42.87	42.87	42.87
a_2_1	211.93	211.93	211.93	80.71	80.71	80.71
a_2_2	6.58	4.92	2.98	27.04	26.33	25.23
a_2_3	6.99	5.90	4.84	70.50	69.21	68.65
a_2_4	41.14	24.47	17.57	52.95	49.09	47.29
a_2_5	16.24	8.88	6.26	34.29	30.57	28.51
b_1	88.97	88.97	88.97	16.93	16.93	16.93
b_2	967.74	491.80	337.08	112.41	107.57	101.55
b_3	65.85	33.89	22.69	6.37	6.24	6.19
b_4	2,360.00	1,475.00	1,072.73	340.64	144.18	114.51
Average	**364.82**	**241.57**	**188.76**	**70.56**	**54.85**	**51.54**

Table 7 Comparison of GeNePi and ϵ-Constraints, the latter running for 10, 20 and 30 days for both number of non-dominated solutions and hypervolume on the different instances. A number lower than 100 means GeNePi is outperformed by ϵ-Constraints.

	GeNePi	ϵ-Constraints					
		10 days		20 days		30 days	
	Time	**Time**	$\frac{\text{Time}}{\#\text{Solutions}}$	**Time**	$\frac{\text{Time}}{\#\text{Solutions}}$	**Time**	$\frac{\text{Time}}{\#\text{Solutions}}$
a_1_1	2	483,747	378	483,747	378	483,747	378
a_1_2	3,106	864,000	23,351	1,728,000	32,000	2,592,000	33,231
a_1_3	441	864,000	10,410	1,728,000	11,368	2,592,000	12,226
a_1_4	309	864,000	10,800	1,728,000	11,597	2,592,000	11,623
a_1_5	332	280,749	5,013	280,749	5,013	280,749	5,013
a_2_1	3,905	267,638	2,455	267,638	2,455	267,638	2,455
a_2_2	600	864,000	289	1,728,000	431	2,592,000	393
a_2_3	695	864000	299	1,728,000	505	2,592,000	621
a_2_4	342	864,000	1,405	1,728,000	1,671	2,592,000	1,800
a_2_5	347	864,000	638	1,728,000	697	2,592,000	737
b_1	14,991	365,675	1,344	365,675	1,344	365,675	1,344
b_2	10,028	864,000	27,871	1,728,000	28,328	2,592,000	29,124
b_3	39,596	864,000	3,512	1,728,000	3,615	2,592,000	3,630
b_4	63,535	864,000	172,800	1,728,000	216,000	2,592,000	235,636

Table 8 Comparison of ϵ-Constraints execution time (s) and average time per solution (s) when run for different periods on the different ROADEF instances, against GeNePi execution time (s).

8 Conclusion

Reassigning processes to servers automatically is complex (a lot of dimensions and constraints), large-scale for most of the real instances (data centres are usually big computing facilities) and needs to consider different objectives.

Multi-objective approaches are good when the set of possible solutions is large and extracting the 'best solution' is difficult. In this case, the system needs to be assisted by decision-makers who can evaluate the different solutions with respect to their value in the different dimensions of the problem. Here,

we defined the machine reassignment in the three-dimensional space defined by: (i) reliability of the assignment, (ii) migration cost of the reassignment, and (iii) energy consumption of the assignment.

In this paper we define the multi-objective machine reassignment problem and compare different algorithms: classical heuristics, metaheuristics and hybrid metaheuristics. In particular GeNePi, a hybrid metaheuristic based on three successive optimisation steps: Ge, a variant of the constructive phase of GRASP, which aims at finding an initial population with solutions representing every objective; Ne, based on a genetic algorithm called NSGA-II that mixes solutions of the initial population and tries to find new solutions (more diverse ones); and Pi a local search that looks for more solutions in the neighbourhood of those that GeNePi has already found.

We showed on a large experimental validation that GeNePi outperforms other non-hybrid algorithms: it finds 4 times more non-dominated solutions that are scattered over more of the search space (hypervolume is more than 100% better) – which is desirable as we want to offer decision-makers a large variety of different solutions. GeNePi also outperforms other hybrid metaheuristics (it finds more than double the amount of non-dominated solutions and achieves a better hypervolume with over 15% on average). A comparison of GeNePi against one of the well-known exact methods for solving multi-objective problems (i.e., ϵ-Constraints) shows that GeNePi gets more than 188% non-dominated solutions and a hypervolume of more than 21% on average than ϵ-Constraints when it is run for 30 days. At the same time, GeNePi succeeds in keeping its execution time relatively low. GeNePi is tens of thousands of times faster on average than ϵ-Constraints, and even faster or on the same order of magnitude as one single mono-objective optimisation of the same problem.

There are three directions that we would like to explore further in the future: (i) the sensitivity of GeNePi to the parameter tuning, (ii) electricity consumption which will need to incorporate more parameters (such as cooling of data centres) and (iii) Service Level Agreements.

9 Acknowledgement

This work was supported, in part, by Science Foundation Ireland (SFI) grant 13/IA/1850 and grants 10/CE/I1855 and 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie).

References

Abdullah Alsheddy and Edward EP K Tsang. Guided pareto local search based frameworks for biobjective optimization. In *CEC*, pages 1–8, 2010.

Eric Angel, Evripidis Bampis, and Laurent Gourves. A dynasearch neighborhood for the bicriteria traveling salesman problem. In *Metaheuristics for Multiobjective Optimisation*, pages 153–176, 2004.

Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *ICSE*, pages 1–10, 2011.

Matthieu Basseur. Design of cooperative algorithms for multi-objective optimization: application to the flow-shop scheduling problem. *4OR*, pages 255–258, 2006.

James Beck and Daniel Siewiorek. Modeling multicomputer task allocation as a vector packing problem. In *ISSS*, page 115, 1996.

Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *FGCS*, pages 755–768, 2012.

Eyal Bin, Ofer Biran, Odellia Boni, Erez Hadad, Eliot K Kolodner, Yosef Moatti, and Dean H Lorenz. Guaranteeing high availability goals for virtual machine placement. In *ICDCS*, pages 700–709, 2011.

Felix Brandt, Jochen Speck, and Markus Völker. Constraint-based large neighborhood search for machine reassignment. *Ann Oper Res*, 242:63–91, 2016.

Franck Butelle, Laurent Alfandari, Camille Coti, Lucian Finta, Lucas Létocart, Gérard Plateau, Frédéric Roupin, Antoine Rozenknop, and Roberto Wolfler Calvo. Fast machine reassignment. *Ann Oper Res*, pages 133–160, 2016.

Alberto Caprara and Paolo Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Appl Math*, pages 231–262, 2001.

European Commission. Datacentre energy efficiency. http://re.jrc.ec.europa.eu, 2007.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *TEVC*, pages 182–197, 2002.

Shyam Kumar Doddavula, Mudit Kaushik, and Akansha Jain. Implementation of a fast vector packing algorithm and its application for server consolidation. In *CloudCom*, pages 332–339, 2011.

Nasim Donyagard Vahed, Mostafa Ghobaei-Arani, and Alireza Souri. Multiobjective virtual machine placement mechanisms using nature-inspired metaheuristic algorithms in cloud environments: A comprehensive review. *International Journal of Communication Systems*, page e4068, 2019.

Emanuel Falkenauer. *Genetic algorithms and grouping problems*. Wiley New York, 1998.

Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *JGO*, pages 109–133, 1995.

D Filani, J He, S Gao, M Rajappa, A Kumar, P Shah, and R Nagappan. Comparing vm-placement algorithms for on-demand clouds. In *Dynamic data center power management: Trends, issues, and solutions*, 2008.

Mark Fleischer. The measure of pareto optima applications to multi-objective metaheuristics. In *EMO*, pages 519–533, 2003.

Michaël Gabay and Sofia Zaourar. A GRASP approach for the machine reassignment problem. In *EURO*, 2012.

Xavier Gandibleux. Peek–shape–grab: A methodology in three stages for approximating the non-dominated points of multiobjective discrete/combinatorial optimization problems with a multiobjective metaheuristic. In *EMO*, pages 221–235, 2017.

Haris Gavranović, Mirsad Buljubašić, and Emir Demirović. Variable neighborhood search for google machine reassignment problem. *ENDM*, pages 209–216, 2012.

Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Comput Oper Res*, pages 347–358, 2004.

Google/ROADEF/EURO CHALLENGE 2012. Google/roadef/euro challenge. http://challenge.roadef.org/2012/en/, 2012.

Ronald L Graham. Bounds on multiprocessing anomalies and related packing algorithms. In *SJCC*, pages 205–217, 1972.

Rodolfo Hoffmann, María-Cristina Riff, Elizabeth Montero, and Nicolas Rojas. Google challenge: A hyperheuristic for the machine reassignment problem. In *CEC*, pages 846–853, 2015.

Klaus Jansen and Sabine Öhring. Approximation algorithms for time constrained scheduling. *Inform Comput*, pages 85–108, 1997.

W Jaśkowski, M Szubert, and P Gawron. A hybrid mip-based large neighborhood search heuristic for solving the machine reassignment problem. *Ann Oper Res*, pages 1–30, 2015.

Gueyoung Jung, Matti Hiltunen, Kaustubh R Joshi, Richard D Schlichting, Calton Pu, et al. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *ICDCS*, pages 62–73, 2010.

Hans Kellerer and Vladimir Kotov. An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing. *Oper Res Lett*, pages 35–41, 2003.

William Leinberger, George Karypis, and Vipin Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *ICPP*, pages 404–412, 1999.

Xi Li, Anthony Ventresque, Nicola Stokes, James Thorburn, and John Murphy. ivmp: an interactive vm placement algorithm for agile capital allocation. In *CLOUD*, pages 950–951, 2013.

Xi Li, Anthony Ventresque, John Murphy, and James Thorburn. A fair comparison of VM placement heuristics and a more effective solution. In *IEEE 13th International Symposium on Parallel and Distributed Computing, ISPDC*, pages 35–42, 2014.

Chia-Hung Lien, Ying-Wen Bai, and Ming-Bo Lin. Estimation by software for the power consumption of streaming-media servers. *TIM*, pages 1859–1870, 2007.

Ramon Lopes, Vinicius WC Morais, Thiago F Noronha, and Vitor AA Souza. Heuristics and matheuristics for a real-life machine reassignment problem. *ITOR*, pages 77–95, 2015.

Yuri Malitsky, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Tuning parameters of large neighborhood search for the machine reassignment

problem. In *CPAIOR*, pages 176–192, 2013.

Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *CSUR*, page 11, 2015.

R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, pages 369–395, 2004.

Renaud Masson, Thibaut Vidal, Julien Michallet, Puca Huachi Vaz Penna, Vinicius Petrucci, Anand Subramanian, and Hugues Dubedout. An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Syst Appl*, pages 5266–5275, 2013.

George Mavrotas. Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Appl Math Comput*, pages 455–465, 2009.

Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Comparing solution methods for the machine reassignment problem. In *CP*, pages 782–797, 2011.

K Mills, J Filliben, and C Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *CloudCom*, pages 91–98, 2011.

Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.

Gabriel M Portal, Marcus Ritt, Leonardo M Borba, and Luciana S Buriol. Simulated annealing for the machine reassignment problem. *Ann Oper Res*, pages 1–22, 2012.

Robin C Purshouse and Peter J Fleming. On the evolutionary optimization of many conflicting objectives. *TEVC*, pages 770–784, 2007.

Takfarinas Saber. Multi-objective virtual machine reassignment for large data centres. 2017.

Takfarinas Saber, Anthony Ventresque, Xavier Gandibleux, and Liam Murphy. Genepi: A multi-objective machine reassignment algorithm for data centres. *HM*, pages 115–129, 2014a.

Takfarinas Saber, Anthony Ventresque, Liam Murphy, and El-Ghazali Talbi. Multi-objective vm reassignment for the enterprise. In *Meta*, 2014b.

Takfarinas Saber, Anthony Ventresque, Ivona Brandic, James Thorburn, and Liam Murphy. Towards a multi-objective vm reassignment for large decentralised data centres. In *UCC*, 2015a.

Takfarinas Saber, Anthony Ventresque, Joao Marques-Silva, James Thorburn, and Liam Murphy. Milp for the multi-objective vm reassignment problem. In *ICTAI*, pages 41–48, 2015b.

Takfarinas Saber, Joao Marques-Silva, James Thorburn, and Anthony Ventresque. Exact and hybrid solutions for the multi-objective vm reassignment problem. *IJAIT*, 2017.

Takfarinas Saber, Delavernhe Florian, Papadakis Mike, O'Neill Michael, and Anthony Ventresque. A hybrid algorithm for multi-objective test case selection. In *CEC*, 2018a.

Takfarinas Saber, James Thorburn, Liam Murphy, and Anthony Ventresque. Vm reassignment in hybrid clouds for large decentralised companies: A multi-objective challenge. *Future Generation Computer Systems*, pages 751–764, 2018b.

Bianca Schroeder and Garth A Gibson. A large-scale study of failures in high-performance computing systems. *TDSC*, pages 337–351, 2010.

Hadas Shachnai and Tami Tamir. Approximation schemes for generalized two-dimensional vector packing with application to data placement. *JDA*, pages 35–48, 2012.

Mark Stillwell, Frederic Vivien, and Henri Casanova. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *IPDPS*, pages 786–797, 2012.

Ayad Turky, Nasser R Sabar, Abdul Sattar, and Andy Song. Parallel late acceptance hill-climbing algorithm for the google machine reassignment problem. In *Australasian Joint Conference on Artificial Intelligence*, pages 163–174, 2016.

Ayad Turky, Nasser R Sabar, Abdul Sattar, and Andy Song. Evolutionary learning based iterated local search for google machine reassignment problems. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 409–421, 2017a.

Ayad Turky, Nasser R Sabar, and Andy Song. Neighbourhood analysis: a case study on google machine reassignment problem. In *Australasian Conference on Artificial Life and Computational Intelligence*, pages 228–237, 2017b.

Ayad Turky, Nasser R Sabar, and Andy Song. Cooperative evolutionary heterogeneous simulated annealing algorithm for google machine reassignment problem. *GPEM*, pages 183–210, 2018.

András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, pages 101–132, 2000.

Marc Visée, Jacques Teghem, Marc Pirlot, and Ekunda L Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, pages 139–155, 1998.

William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom*, pages 254–265, 2009.

Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. In *ICAC*, pages 225–234, 2011.

Jing Xu and Jose AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *GreenCom*, pages 179–188, 2010.

Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms — a comparative case study. In *PPSN*, pages 292–301, 1998.

Eckart Zitzler, Marco Laumanns, Lothar Thiele, Carlos M Fonseca, and Viviane Grunert da Fonseca. Why quality assessment of multiobjective optimizers is difficult. In *GECCO*, pages 666–673, 2002.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *TEVC*, pages 117–132, 2003.