Exact and Hybrid Solutions for the Multi-objective VM Reassignment Problem

Takfarinas Saber[1, ✉], Joao Marques-Silva[2], James Thorburn[3] and Anthony Ventresque[1, 4, ✉]

[1] *Lero@UCD, School of Computer Science, University College Dublin, Ireland*
Email: takfarinas.saber@ucdconnect.ie, anthony.ventresque@ucd.ie
[2] *Instituto Superior Técnico, Lisboa, Portugal, Email: jpms@ist.utl.pt*
[3] *IBM Software Group, Toronto, Canada, Email: jthorbur@ca.ibm.com*
[4] *IBM Research, Damastown Industrial Estate, Dublin, Ireland*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Machine Reassignment is a challenging problem for constraint programming (CP) and mixed integer linear programming (MILP) approaches, especially given the size of data centres. Hybrid solutions mixing CP and heuristic algorithms, such as, large neighbourhood search (CBLNS), also struggle to address the problem given its size and number of constraints. The multi-objective version of the Machine Reassignment Problem is even more challenging and it seems unlikely for CP, MILP or hybrid solutions to obtain good results in this context. As a result, the first approaches to address this problem have been based on other optimisation methods, including metaheuristics. In this paper we study three things: (i) under which conditions a mixed integer optimisation solver, such as *IBM ILOG CPLEX*, can be used for the Multi-objective Machine Reassignment Problem; (ii) how much of the search space can a well-known hybrid method such as CBLNS explore; and (iii) can we find a better hybrid approach combining MILP or CBLNS and another recent metaheuristic proposed for the problem (GeNePi). We show that MILP can handle only small or medium scale data centres, and with some relaxations, such as, an optimality tolerance gap and a limited number of directions explored in the search space. CBLNS on the other hand struggles with the problem in general but achieves reasonable performance for large instances of the problem. However, we show that our hybridisation improves both the quality of the set of solutions (CPLEX+GeNePi and CBLNS+GeNePi improve the solutions by +17.8% against CPLEX alone and +615% against CBLNS alone) and number of solutions (8.9 times more solutions than CPLEX alone and 56.76 times more solutions than CBLNS alone), while the processing time of CPLEX+GeNePi and CBLNS+GeNePi increases only by 6% and 16.4% respectively. Overall, the study shows that CPLEX+GeNePi is the best algorithm for small instances (CBLNS+GeNePi only gets 45.2% of CPLEX+GeNePi's hypervolume) while CBLNS+GeNePi is better than the others on large instances (that CPLEX+GeNePi cannot address).

Keywords: Multi-objective Optimisation; VM/Machine Reassignment; Mixed Integer Linear Programming; Large Neighbourhood Search; Hybrid-Metaheuristics.

1. Introduction

Background and Research Challenge

Optimisation of data centres, through reassignment of virtual machines (VMs), is largely seen as one of the biggest challenges in data centre management [1]: not only servers are underutilised [2] and the potential savings are important, but the problem has a lot of constraints

and is difficult to solve [3]. Constraint programming (CP) and mixed[a] integer programming (MILP) are known to be inefficient for such large scale problems with a limited execution time [4,5], and usually researchers focus on other optimisation techniques (e.g., local search [6,7] or greedy algorithms [8]) or mix CP or MILP with some other optimisation solutions (e.g., local search [5]) – but often with little or no success. For instance, authors in [5] show that a solution based on Large Neighbourhood Search (LNS), while good compared to the other state-of-the-art methods, struggles to improve the quality and quantity of solutions.

In this paper, we address a slightly different and more relevant problem for the industry: *the Multi-objective VM Reassignment Problem*. 'Optimising a data centre' seems to suggest that there is something like a *best* reassignment of VMs, but in an enterprise there is no best placement: it is all about which objectives are favoured (by whom and when). It is not hard to imagine that different capital allocators (CA, the managers of data centres) may have different perspectives on the best way of making the system better. For instance some CAs may consider that energy consumption is the most important element, while for others it can be the cost of licensing; or some CAs see the reliability as the key element (for instance if they run critical applications), while other CAs have a strict policy regarding response time and then collocation of VMs. These preferences, or policies, may evolve or be in competition: for instance if CAs have two policies for their data centre (e.g., "electricity has to decrease by $x\%$" and "management cost has to be limited to $y\%$") and some reassignment solutions can serve both, which one to favour? In this context, CP, MILP or LNS do not seem promising approaches, as the search space is large and the constraints hard and complex. As far as we know, the only related work tackles the problem using some other optimisation techniques. 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 CAs cannot wait more than a few hours or days: they verify and modify the solutions to suit their needs before making any decision. In this paper, as it is commonly accepted by practitioners and in the literature [5,9] we use a time limit for the Multi-objective Reassignment Problem.

However, given that CPLEX' solutions are generally of better quality than the ones of other optimisation techniques, and that there are several relaxation mechanisms in CPLEX and the multi-objective problem itself, we think in this paper that it is important to study whether a MILP solver, such as CPLEX, can be used for the Multi-objective Machine Reassignment Problem. On the other hand, while LNS is not suited for such (very) large Multi-objective problems, there are ways to limit the search space (e.g., exploring only a few directions) that we believe are worth exploring in this paper. Eventually, we also investigate whether a hybrid method combining CPLEX or CBLNS and a metaheuristic could lead to improved results, the intuition being that mixing the good aspects of each could help to solve this complex problem.

[a]Note that while the constraints can be expressed by binary variables, the objectives (especially the reliability cost, see Section 3) require real variables.

Example

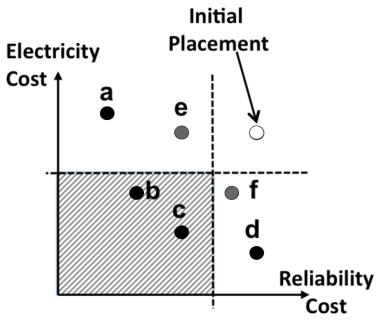


Figure 1. Possible reassignments in 2-dimensions (in black, the good ones) and an organisation's policies (area in grey).

As a motivating example, consider Figure 1 that shows a reassignment problem with two objectives: the electricity consumption and the reliability (the lower the better for both). Points a to f are all the possible reassignments, satisfying all the constraints of the data centre. The good reassignments, i.e., those that are better than any other in a particular combination of objectives, are in black. Solution e, for instance, is not a good reassignment as it is worse than b on both objectives - same applies to f wrt. c. The decision makers may define some policies, i.e., some rules that describe the desired optimisations: here a maximal value for the electricity consumption and the reliability, defining the area of good assignments that the decision makers can select (grey area on Figure 1). There are still two possible reassignments, b and c. The decision makers can then evaluate them locally and make a relative decision among them, such as "c gives me a better electricity cost but it is not a huge gain, while b's reliability is higher – so we favour b". Any technique addressing this problem needs to explore the search space in all directions in order to find varied and good solutions – which is expensive given the size of the problem.

Contributions

In this paper we make the following contributions: (i) we give a thorough study of how suited is a MILP solver (CPLEX) for the Multi-objective VM Reassignment Problem (the problem definition is given in Section 3). We show that it is useful only for rather small or medium scale data centres and with some relaxations: a certain tolerance gap and a limited number of directions explored in the search space (Section 5); (ii) we give another exhaustive study of whether CBLNS (an approach based on LNS) can solve the Multi-objective VM Reassignment Problem and what parameters give the best solution. We show that there is an optimal number of directions that we can explore in the search space (Section 6); (iii) we propose an algorithm based on the combination of CPLEX or CBLNS and a hybrid-metaheuristic to improve the performance both quantitatively and qualitatively,

while keeping the execution time acceptable (Section 7).

Hybridisation is not a novel idea as such [10], however combining a solver (which aim is to find 'proper' solutions and not only to relax the problem) and a metaheuristic to benefit from both worlds in a multi-objective context is novel to our best knowledge. CP and MILP provide better solutions (when they get any, as they are expensive/slow) so our goal here is to help CPLEX and CBLNS as much as we can (i.e., increasing the optimality gap and varying the number of vectors for CPLEX, and varying the number of vectors for CBLNS) to get some solutions to feed in the metaheuristic to cover the search space. Note that Mehta et al. [5] use CP on a relaxed problem, not the original VM placement one, which leads us to think that CP is inefficient for our problem – and anyway we also tried ILOG CP Optimizer and noticed extremely poor performances.

2. Related Work

In this section we study the related work in the area relevant to our approach: *d − Dimensional* Bin Packing, VM Reassignment and Multi-objective VM Reassignment.

2.1. *d − Dimensional Bin Packing*

The *d − Dimensional* Bin Packing can be defined as packing a set of objects characterised by a vector of resource demands, in the least number of homogeneous bins. This problem has two main branches depending on whether it concerns a geometrical (with rotations in the space) or vectorial (following the dimension axes) packing. The first branch is out of scope of this work and we only focus on the latter. The vectorial packing –on its own– has a large number of applications such as multi-processors going back to the early seventies [11].

In the context of two dimensional bin packing, Caprara and Toth [12] developed a set of heuristics and exact solutions, which were since outperformed by an approximation algorithm proposed by Kellerer and Kotov [13] with a performance ratio of 2 in the worst-case.

Concerning problems with multiple resources, Becks et al. [14] modelled and made a thorough evaluation of different algorithms for assigning tasks in a multi-computer. Whereas Leinberger et al. [15] addressed a same problem, but for systems running tasks in parallel.

Some works add different constraints to the tradition *d − Dimensional* Bin Packing. Having conflicting items is the most related to our work. Jansen and Öhring [16] studied the partitioning of conflicting objects over independent sets and proposed some approximation algorithms, while Gendreau et al. [17] put forwards several heuristics and lower bounds.

Most Recent works deal with the vectorial *d − Dimensional* Bin Packing for storing multi-media content. They either study different algorithms (e.g., heuristics from the First-Fit family [18]) or try to find a better approximation algorithm [19].

2.2. *VM Reassignment Problem*

While the *d − Dimensional* Bin Packing aims at reducing the number of bins, the VM reassignment problem considers moving items between an already given set of bins (a.k.a., machines). It tries to optimise different goals while integrating many topic-related constraints.

In the context of managing resources in cloud computing environments, several optimisation problems have been defined and studied. Doddavula et al. [20] worked on optimising resource utilisation by reassigning tasks to different machines by comparing commonly used First Fit algorithms. Beloglazov et al. [1] worked on the reduction of cloud computing data centres' energy footprint and proposed resource allocation heuristics to tackle this problem. Stillwell et al. [21] pushed the boundaries further by considering a resource allocation with heterogeneous machines.

Due to the increasing popularity and scale of the cloud computing, Google (one of the public cloud big players with Google Cloud Engine) proposed a challenge at the ROADEF/EURO 2012 [9] forum with a detailed formulation for the problem called Machine Reassignment Problem (MRP) and provided real life large scale instances. The challenge attracted many participants, with algorithms of different types. Although the majority were based on a local search (e.g., Local Search (LS [6]), Large Neighbourhood Search (LNS [7]) or Multi-Start Local Search (MS-LS [22])), there were others based on Greedy Randomized Adaptive Search (GRASP [8]), Simulated Annealing (SA [23]), or a combination of either CP or MILP solvers, with some other optimisation solutions (e.g., Local Search with either a CP [5], or a MILP [24] solver).

2.3. *Multi-Objective VM Reassignment Problem*

The multi-objective consolidation (reassignment) of Virtual Machines in a data centre has been described recently as an important research challenge for data centres [1]. Although we can find many works dealing with this problem in the literature [25,26], 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 [27] or limiting their study to only a bi-objective resolution [28]).

The Multi-Objective VM Reassignment Problem 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 [29] was quite recent. A linear formulation for the same multi-objective problem and a study of the usability of a MILP solver were put forward [30], but the work was only limited to the smallest instances. Whereas, we present in this work a new (and thorough) study of CBLNS for the Multi-objective VM Reassignment Problem and we propose new hybrid solutions and their evaluation. A formulation similar to ours has also been studied in the context of decentralised data centres [31,32], which considers that each site is independent when making its own machine placement. The study provides a full multi-objective model for the VM reassignment in such infrastructures and conducts a thorough evaluation of several algorithms.

3. Problem Definition

In this section, we give a multi-objective extension of the VM Reassignment Problem originally proposed by Mehta et al. [5] and inspired from Google/ROADEF [9] (see Section 4.1 for more details about this challenge). We first describe briefly the different elements of data

centres, followed by the constraints of the problem, and we finish with the different objectives that we believe are the most relevant (note that this approach is agnostic to objectives and would work with any other linear objective function).

3.1. *Problem Description and Notation*

A data centre is composed of a set $\mathcal{M}$ of physical machines (PMs or servers). Each machine $m_i \in \mathcal{M}$ has a finite amount $Q_{m_i,r}$ of resource $r \in \mathcal{R}$ (e.g., CPU, RAM, storage). Machines $m_i \in \mathcal{M}$ belonging to a same rack are linked with high network connections, and thus considered as being in the same neighbourhood $N(m_i) \in \mathcal{N}$. Resources are of two different types: (i) transient resources ($r \in \mathcal{TR} \subseteq \mathcal{R}$, e.g., RAM or storage) that are consumed at the original host and also at the destination one during a migration process, or (ii) non-transient ($r \in \overline{\mathcal{TR}}$, e.g., CPU). The data centre is in charge of a set of VMs $v \in \mathcal{V}$ with resource requirements $d_{v,r}$ for every $r \in \mathcal{R}$. $M_0(v)$ and $M(v)$ respectively indicate the initial and the final host of the VM v during the reassignment. VMs composing a same multi-tier application are usually gathered by services $\mathcal{S} = \{s_1, \ldots, s_p\}$, with $s_p = \{v_p^1, \ldots, v_p^q\}$.

3.2. *Constraints*

We present here the linear constraints of our problem. We follow the problem's linear constraints provided in Mehta et al. [5].

3.2.1. *Reassignment Constraints*

Consider a binary variable $x_{v,m}$ for every VM $v \in \mathcal{V}$ and for each machine $m \in \mathcal{M}$, which is set to 1 if $M(v) = m$ and 0 otherwise. Constraints (1) ensure that every VM is reassigned to one and only one machine:

$$\forall v \in \mathcal{V}, \quad \sum_{m \in \mathcal{M}} x_{v,m} = 1 \tag{1}$$

3.2.2. *Capacity Constraints*

There are two ways of computing resource utilisation $U_{m,r}$ of a machine $m \in \mathcal{M}$ and a resource $r \in \mathcal{R}$: (2) for non-transient resources and (3) for transient resources.

$$\forall r \in \overline{\mathcal{TR}}, \ \forall m \in \mathcal{M}, \quad U_{m,r} \ = \ \sum_{v \in \mathcal{V}} d_{v,r}.x_{v,m} \tag{2}$$

$$\forall r \in \mathcal{TR}, \forall m \in \mathcal{M}, U_{m,r} = \sum_{v \in \mathcal{V} | M_0(v) = m} d_{v,r}.x_{v,m} + \sum_{v \in \mathcal{V} | M_0(v) \neq m} d_{v,r}.x_{v,m} \tag{3}$$

The total resource utilisation of a machine $m \in \mathcal{M}$ should not exceed its capacity $Q_{m,r}$ for every $r \in \mathcal{R}$:

$$\forall r \in \mathcal{R}, \ \forall m \in \mathcal{M}, \quad U_{m,r} \leq Q_{m,r} \tag{4}$$

3.2.3. *Conflict Constraints*

VMs which belong to a same service have to be reassigned to different machines (e.g., for replication purposes).

$$\forall s \in \mathcal{S},\ \forall m \in \mathcal{M},\ \sum_{v \in s \mid M(v)=m} x_{v,m} \leq 1 \tag{5}$$

3.2.4. *Dependency Constraints*

Services sometimes depend on each other. VMs of these services need to be close to each other in order to achieve high performance (e.g., in the case of multi-tier applications). Let $\mathcal{D}$ be the set of service dependencies such that $\mathcal{D} = \{(s_i, s_j) | s_i, s_j \in \mathcal{S}$ and s_i depends on $s_j\}$, then:

$$\forall s_i, s_j \in \mathcal{S}, (s_i, s_j) \in \mathcal{D} \implies \forall v_a \in s_i, \exists v_b \in s_j | N(M(v_a)) = N(M(v_b)) \tag{6}$$

To give a linear definition of constraints (6), we introduce the binary variables $y_{s,n}$ for every service $s \in \mathcal{S}$ and for each neighbourhood $n \in \mathcal{N}$. Constraints (7) and (8) ensure that each variable $y_{s,n}$ is set to 1 if at least one VM from the service s is hosted by a machine in the neighbourhood $n \in \mathcal{N}$ and to 0 otherwise:

$$\forall s \in \mathcal{S},\ \forall n \in \mathcal{N} \quad \sum_{v \in s} \sum_{m \in n} x_{v,m} \leq |\mathcal{N}|.|\mathcal{S}|.y_{s,n} \tag{7}$$

$$\forall s \in \mathcal{S},\ \forall n \in \mathcal{N} \quad \sum_{v \in s} \sum_{m \in n} x_{v,m} \geq y_{s,n} \tag{8}$$

If a service s_i depends on s_j, constraints (9) guarantee that there is not any VM from s_i reassigned to a machine in a neighbourhood $n \in \mathcal{N}$ without having at least one VM from s_j reassigned to that neighbourhood:

$$\forall (s_i, s_j) \in \mathcal{D},\ \forall n \in \mathcal{N} y_{s_i,n} \leq y_{s_j,n} \tag{9}$$

3.2.5. *Spread Constraints*

Data centres are often decentralised and we assume here that they are composed of different sites $\mathcal{L}$, each machine belonging to a location $l \in \mathcal{L}$. For reliability and security reasons, some services need to be spread over at least *spreadNumber* locations. Let us introduce a binary variable $z_{s,l}$ for every service $s \in \mathcal{S}$ and for each location $l \in \mathcal{L}$. Constraints (10) and (11) ensure that $z_{s,l}$ gets the value 1 if the service s has at least one VM running in a machine at the location l:

$$\forall s \in \mathcal{S},\ \forall l \in \mathcal{L} \quad \sum_{v \in s} \sum_{m \in l} x_{v,m} \leq |\mathcal{N}|.|\mathcal{S}|.z_{s,l} \tag{10}$$

$$\forall s \in \mathcal{S},\ \forall l \in \mathcal{L} \quad \sum_{v \in s} \sum_{m \in l} x_{v,m} \geq z_{s,l} \tag{11}$$

Constraints (12) force every service s to run on at least $spreadNumber_s$ number of locations:

$$\forall s \in \mathcal{S}, \quad \sum_{l \in L} z_{s,l} \geq spreadNumber_s \tag{12}$$

Machine Reassignment An assignment A of VMs to machines is a mapping: $A : \mathcal{P} \mapsto \mathcal{M}$, such that $A(v, \mathcal{M}) \rightarrow m$, which satisfies the constraints (1–5 and 7–12).
A reassignment is a function: $ReA : A \mapsto A$ which returns a new assignment for a given initial assignment of VMs to machines.

3.3. *Objectives*

We focus here on three objectives: electricity cost, VM migration cost and reliability cost, as they are recognised in the literature [33,34,35] and make sense in practice. The multi-objective variant of the Machine Reassignment Problem (see Definition above) consists in minimising the cost functions defined by the objectives.

3.3.1. *Reliability Cost*

For each machine $m \in \mathcal{M}$ and each resource $r \in \mathcal{R}$, we define the safety capacity $SC_{m,r}$ as the amount of resource that is 'safe' to allocate without overloading m – this is similar to the resource buffer that placement algorithms often assume [36]. The risk of failure $R_{m,r}$ is then the difference between the actual utilisation of resource r and the safety capacity, and the reliability cost R_{cost} represents the 'non reliability' over the full data centre.

$$\forall m \in \mathcal{M},\ \forall r \in \mathcal{R}, \quad R_{m,r} \geq U_{m,r} - SC_{m,r} \geq 0 \tag{13}$$

$$R_{cost} = \sum_{r \in R} \sum_{m \in M} R_{m,r} \tag{14}$$

3.3.2. *Electricity Cost*

For each machine $m \in \mathcal{M}$, we introduce a binary variable o_m to be set by constraints (15) to 1 if the machine m is switched on (i.e., hosts at least one VM) and 0 otherwise.

$$\forall m \in \mathcal{M}, \quad o_m \leq \sum_{v \in V} x_{v,m} \leq |\mathcal{V}|.o_m \tag{15}$$

The electricity cost E_{cost} is composed of the electricity consumption of each machine $m \in \mathcal{M}$ multiplied by its price γ_m. The electricity consumption of a machine m is often

modelled as a linear function of its CPU utilisation [37,38], with α_m being its electricity consumption at idle and β_m its consumption per unit of CPU usage.

$$E_{cost} = \sum_{m \in M} \gamma_m \left(\alpha_m . o_m \quad + \quad \beta_m . U_{m,CPU} \right) \tag{16}$$

3.3.3. *Migration Cost*

For each VM $v \in \mathcal{VM}$, we introduce a binary variable mig_v to be set by constraints (17) to 1 if v is reassigned and 0 otherwise.

$$\forall v \in \mathcal{V}, \sum_{m \in M \setminus M_0(v)} x_{v,m} \; = \; mig_v \tag{17}$$

The migration cost M_{cost} concerns all migrating VMs. For each migrating VM $v \in \mathcal{V}$, the migration cost is the time it takes to: (i) prepare the VM $\mu_1(v)$, (ii) to transfer its image from its initial placement to its new host m $\mu_2(v, M_0(v), m)$ and (iii) to deploy it in the new host $\mu_2(v)$ [34]:

$$M_{cost} = \sum_{v \in V} \left(\left[\mu_1(v) + \mu_3(v) \right] . mig_v \quad + \quad \sum_{m \in \mathcal{M}} \mu_2(v, M_0(v), m) . x_{v,m} \right) \tag{18}$$

Note that our model is based on what was given to participants to the Google/ROADEF challenge [9] (see below Section 4.1 for more details). We do not claim to be exhaustive or even totally accurate and we refer the readers to [39,40,41] for more details on migration costs. Here we only assume that migration has a non-negligible cost in data centres and we use the model and values given by the challenge mentioned above.

4. Experimental Setup

In this section we describe the instances of the problem (data centres to optimise) and the metrics used to judge the proposed systems. We adapted a well-known optimisation dataset [9] to fit our problem. All the algorithms described below have been developed in C++. Experiments were done on one node of a super computer with a 24 core 2.0GHz Intel Ivy Bridge CPU and 128GB of RAM.

4.1. *Dataset*

Google and the ROADEF society (i.e., the French Operations Research society) released a few years ago a dataset, now widely used in the Operations Research community, for the evaluation of VM reassignment solutions [9]. This dataset represents various data centres, of different sizes and characteristics (e.g., various number of resources), with a large number of constraints. This dataset does not provide a multi-objective formulation though and we had to adapt the instances (note that the participants of the challenge optimised only one

single weighted-sum of the costs proposed – hence there is no possible comparison between our work and others using ROADEF). Our instances aim to model realistic scenarios as we observe them in large companies. For our evaluation, we take the 14 first instances, leaving out only the largest ones (see Table 1). Two of the objectives we define are present in ROADEF as cost functions: safety/reliability and migration, and we add electricity. We randomly generate electricity consumption constants (α, β) for every machine $m \in \mathcal{M}$ and also the electricity cost γ for every location $l \in \mathcal{L}$. The dataset also comes with a time limit representing the maximum time allowed for the resolution of the instance (300 seconds). We changed the time limit to: 30s for a_1_1, 1h for $a\_1\_\{2-5\}$, 2h for $a\_2\_\{1-3\}$, and 10h for the other instances, which is considered realistic by large companies in the context of optimisation performed on a regular basis, e.g., monthly or quarterly.

Table 1. Characteristics of the different instances used in our evaluation.

Instance	# Resources	# Machines	# Services	# VMs
a_1_1	2	4	79	100
a_1_2	4	100	980	1,000
a_1_3	3	100	216	1,000
a_1_4	3	50	142	1,000
a_1_5	4	12	981	1,000
a_2_1	3	100	1,000	1,000
a_2_2	12	100	170	1,000
a_2_3	12	100	129	1,000
a_2_4	12	50	180	1,000
a_2_5	12	50	153	1,000
b_1	12	100	2,512	5,000
b_2	12	100	2,462	5,000
b_3	6	100	15,025	20,000
b_4	6	500	1,732	20,000

4.2. *Metrics*

The comparison of algorithms optimising a multi-objective problem is complex as their results can be evaluated from different perspectives, such as: the number of solutions on the Pareto front, the variety of solutions [42], or the spread of these solutions. The comparison is even more complex in our case as the problem is large, the Pareto front is not known in most cases and objectives cannot be considered separately. In this paper, we only consider unary metrics, i.e., returning a single value whatever the number of solutions returned by the algorithms. It makes the comparison of a panel of algorithms easier.

4.2.1. *Number of solutions*

We use the number of *non-dominated (efficient) solutions* in our experiments as our first metric. We refer to it as the quantity of found solutions. This metric is highly important for data centre capital allocators as it gives them more choices. It also provides them with

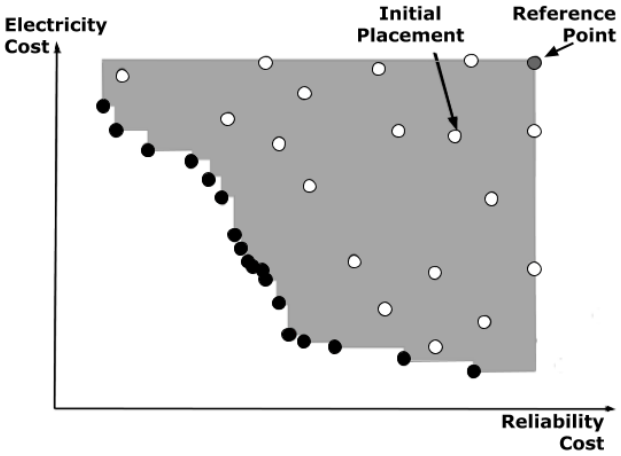


Figure 2. Metrics: number of Pareto solutions (black dots) and hypervolume (grey area).

backup solutions if the preferred choice appears difficult or technically impossible to implement.

4.2.2. *Hypervolume*

The *hypervolume* [43] (a.k.a., $\mathcal{S}$ metric) refers to the quality of a set of solutions. The hypervolume is used [44] in the multi-objective optimisation community for comparing different sets of non-dominated solutions. For every set of solutions, this metric measures the space between the efficient solutions and a reference point far from them. The reference point is defined as a point in the space having the worst objective values and must be identical for all the algorithms, but may be different for every instance. Figure 2 shows a two-dimensional reassignment, with the non-dominated solutions in black and the other (not interesting) solutions in white. The initial placement is one of the solutions, generally not on the Pareto front. The hypervolume is the grey area in this 2D space.

5. CPLEX for the VM Reassignment Problem

The goal of this section is to study both performance and scalability of a MILP solver (i.e., CPLEX) on the Multi-objective VM Reassignment Problem. First, we explore how CPLEX performs on the (Mono-objective) VM Reassignment Problem, i.e., the original problem from ROADEF. We show how difficult the problem is for CPLEX, even in this simpler version. Then we explore the performance of CPLEX for the Multi-objective VM Reassignment and show that it can be tackled under some restrictions, such as, a limited number of directions (*vectors*) explored in the search space and an optimality tolerance gap.

5.1. *CPLEX for the Mono-objective VM Reassignment Problem*

Portal et al. [23] show that the VM Reassignment Problem is too difficult for a MILP solver like CPLEX when applied on the ROADEF instances. CPLEX could only solve 3 instances out of the 14 within the time limit (300s) fixed during the challenge. However CPLEX allows defining an *optimality gap tolerance*, a trigger that stops CPLEX when the current feasible solution falls within a certain percent of CPLEX' best estimation of the lower bound (a value smaller or equal to the actual optimal value). For instance, a 5% gap means that any solution that is 5% away from the estimated lower bound is accepted and stops CPLEX. In addition, we have here a larger time budget to solve the instances.

Table 2 shows the execution time (in seconds) of CPLEX for the resolution of one single vector (the identity vector, i.e., with weights equal 1 for the three objectives). As a reminder, we also add the time limit (last column) that we set for each instance. First, we notice that CPLEX only solves instances a_1_1 and a_1_5 in the time limit (note that 0.01% is the default tolerance gap for CPLEX). This supports the general claim that a MILP solver is inefficient for our problem, even in this simple case with only one vector. We then notice that CPLEX gets a solution with a gap of 0.5% for all small instances, 10% for all medium instances (1% or 5% for some), and solves only one of the hard instances (b_1, with a 5% gap), even with a 50% gap. We also observe that often CPLEX finds a first good solution (e.g., a_1_2, a_1_3 and a_1_4 have a solution for 5% quickly, as evidenced by the same time for 50%, 20%, 10% and 5%) but it is then difficult for CPLEX to improve it. As a conclusion, CPLEX does not seem able to scale to large instances but with a proper gap CPLEX finds good solutions.

Table 2. Execution time (s) of CPLEX for the resolution of the identity vector (all objectives have weights equal 1) depending on the optimality tolerance gap (measured in % from the lower bound).

	50%	**20%**	**10%**	**5%**	**1%**	**0.5%**	**0.1%**	**0.05%**	**0.01%**	**Time (s)**
a_1_1	0.08	0.08	0.08	0.08	0.08	0.12	0.14	0.23	0.25	30
a_1_2	186	183	185	185	1,490	–	–	–	–	3,600
a_1_3	27	27	27	37	625	1,691	–	–	–	3,600
a_1_4	50	50	51	51	98	1,682	–	–	–	3,600
a_1_5	9	9	9	9	9	9	10	19	26	3,600
a_2_1	54	55	159	3,670	–	–	–	–	–	7,200
a_2_2	2,511	2,580	2,580	2,736	–	–	–	–	–	7,200
a_2_3	71	71	71	71	5,816	–	–	–	–	7,200
a_2_4	20,445	20,502	20,655	–	–	–	–	–	–	36,000
a_2_5	21,877	22,492	22,513	–	–	–	–	–	–	36,000
b_1	3,482	6,913	6,913	7,094	–	–	–	–	–	36,000
b_*	–	–	–	–	–	–	–	–	–	36,000

Note: The symbol '–' means that no solution was found in the time limit. The row b_* is for instances b_2, b_3 and b_4.

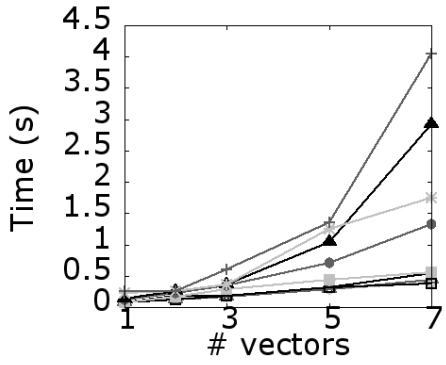
5.2. *CPLEX for the Multi-objective VM Reassignment Problem*

Once we know that CPLEX finds it difficult to solve one weight vector in the search space, we would like to explore what needs to be relaxed in order to help CPLEX optimise more

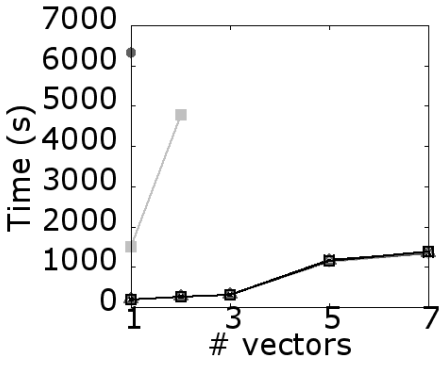
vectors and hence address a proper multi-objective problem. In the current section we look at three elements: (i) given that optimising one weight vector is already difficult to CPLEX and the more vectors we optimise the more we explore the search space, what is the most reasonable number of weight vectors to be optimised? (ii) getting a tiny optimality gap increases exponentially the execution time, therefore what is the best value for this parameter? (iii) what is the best way to use CPLEX? Would it be better to collect all intermediate (feasible) solutions found by CPLEX instead of only keeping the best ones for each weight vector?

Figures 3 show the execution time CPLEX needs to reach different optimality gaps given several maximally spread weight vectors. These vectors are built on the assumption that to explore a maximum of the space the solver needs to target widely spread directions. In our 3-dimensional space (3 objectives), we successively use these vectors: (1,1,1), (0.6, 0.3, 0.1), (0.3, 0.1, 0.6), (0.1, 0.6, 0.3), (0.45, 0.45, 0.1), (0.45, 0.1, 0.45) and (0.1, 0.45, 0.45). We notice that running different vectors increases the final execution time – as it can be expected. It also confirms what we have seen earlier in Table 2 that for large optimality gaps, there is no large difference in terms of execution time (CPLEX quickly finds good solutions), however the more we decrease the gap, the more important is the increase in execution time. We also see that due to the time budget limitation, we cannot run all the possible optimisations with the different vectors for some optimality gaps (e.g., for a_1_2 we could only use 2 vectors out of the 7 possible ones with an optimality gap of 1%). Therefore, a decision has to be made on which values should be set for both variables: the optimality gap and the number of vectors.

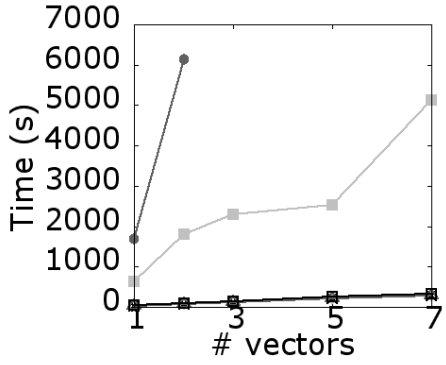
According to Figures 3, two patterns emerge: (i) on small instances: asking CPLEX for an optimality gap smaller than 5% increases its execution time drastically, and (ii) on medium instances: this gap drops to 5 – 10%. Therefore, to keep our optimisation in a reasonable execution time, the larger/more complex is the instance the larger the optimality gap we consider. Unlike what we might think, CPLEX' execution time is very heterogeneous and varies a lot from a vector to another (execution time curves are not linear). Thus, knowing CPLEX' execution time on the first vector does not give any indication/prediction on the execution time for other vectors. Because of the lack of knowledge of the execution time, we have to restrict the number of vectors as much as possible. We are even more constrained regarding some instances such as a_2_4 and a_2_5 where we could only run CPLEX on one vector.



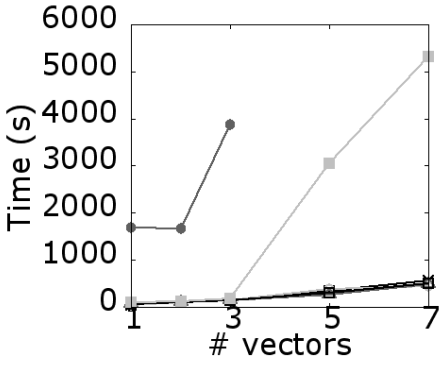
a_1_1



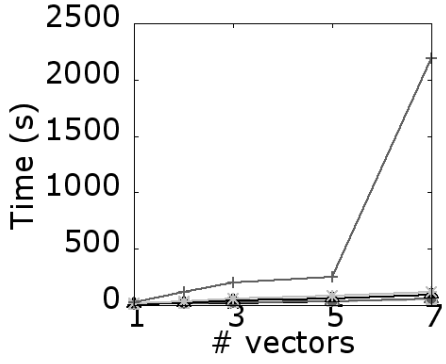
a_1_2



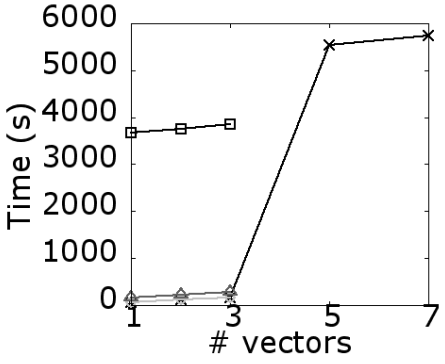
a_1_3



a_1_4



a_1_5



a_2_1

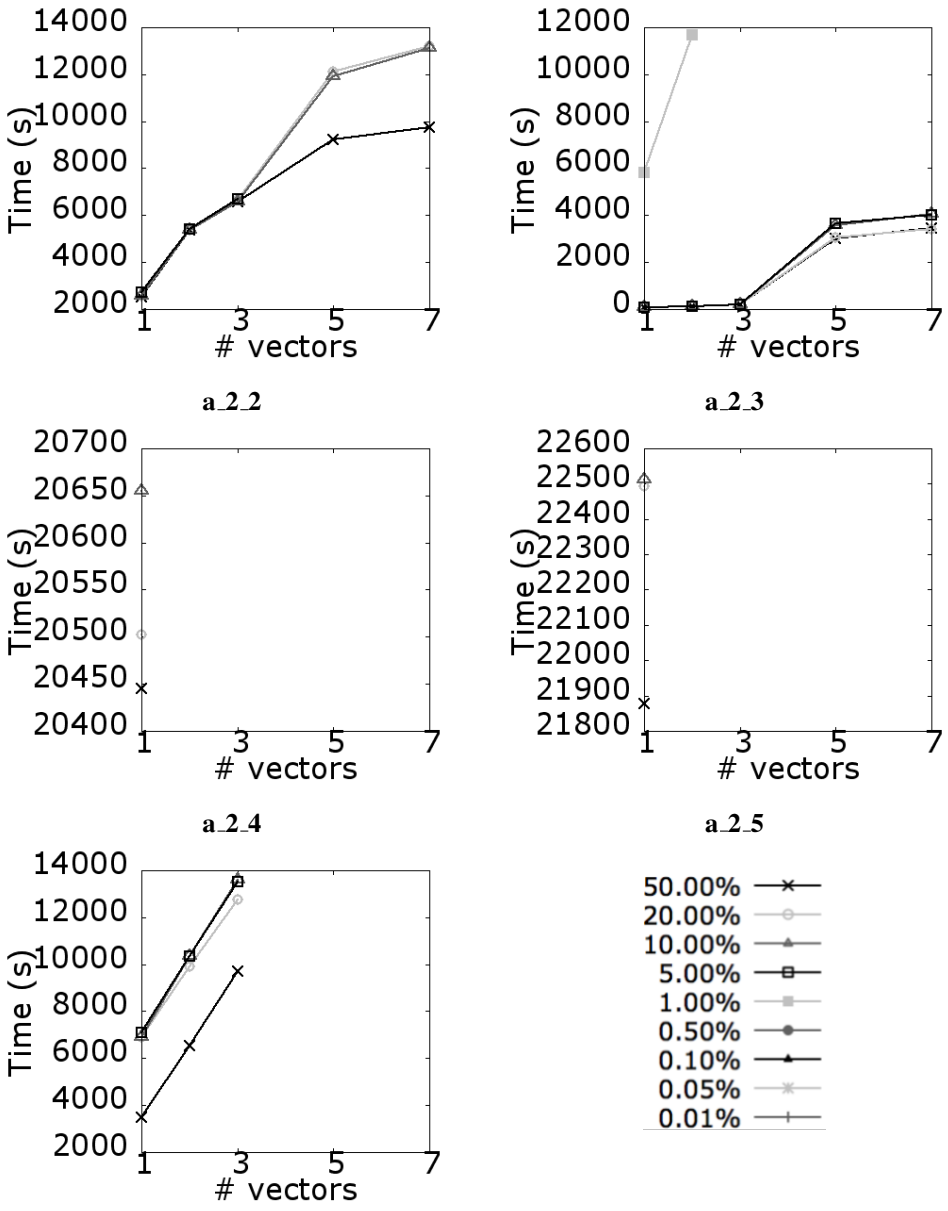
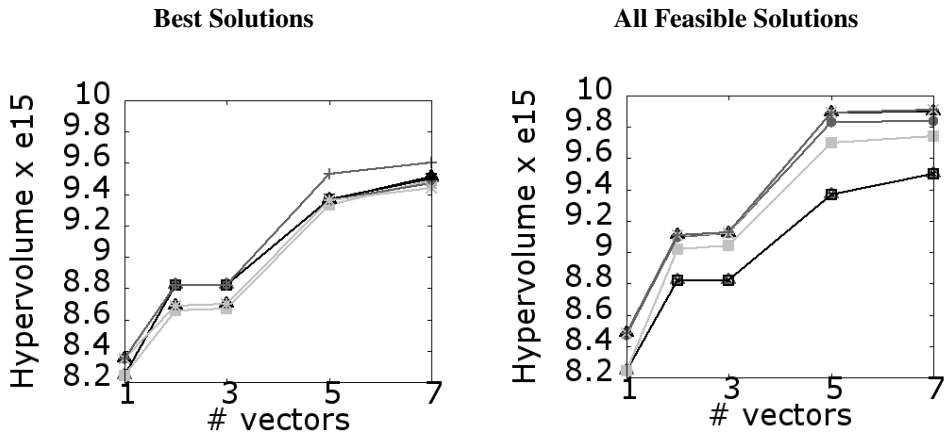


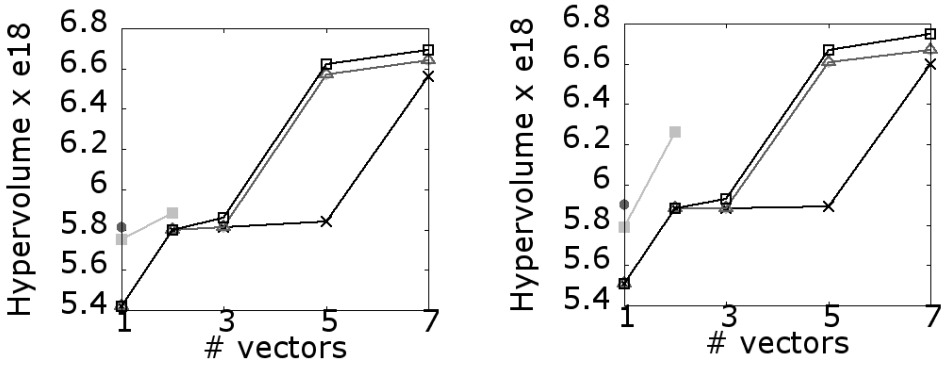
Figure 3: Execution time (s) of CPLEX on ROADEF instances, with maximally spread objective weight vectors and using different optimality gaps.

Another element of the resolution process that might be interesting to look at is the intermediate (feasible) solutions generated by CPLEX. The way CPLEX works is iterative: it first finds a feasible solution that is either discarded/improved if it is not optimal or kept if it is optimal. In our multi-objective context, those intermediate solutions, while not optimal in a particular combination of objectives (remember that CPLEX solves vectors of weights for the objectives), may sometimes carry some interesting reassignments of the VMs, for instance wrt. some single objective – and hence improve the hypervolume.

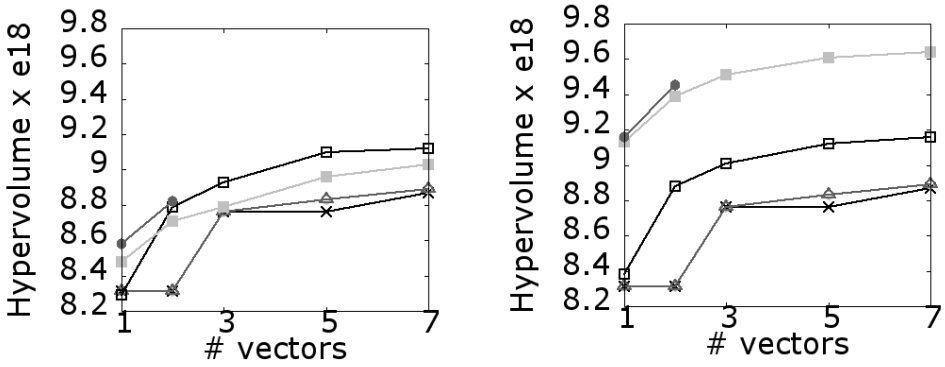
Figures 4 show the hypervolume obtained on the different instances by CPLEX when increasing the number of vectors for different optimality gaps going from 50% to 0.01%. The figures at the left side show the hypervolume obtained when only the best solutions are kept in the solution set while the figures at the right side give the hypervolume for the same experiments with all solutions in the solution set. We notice from the graphs at the left that optimising the objectives using a vector of weights with a small gap does not imply getting a better hypervolume for the multiple objectives (e.g., on the instance a_1_3, two vectors and a gap of 5% get a better hypervolume than using a gap of 1%). This is mainly due to the fact that optimising a compromise of objectives using their linear combination may lead to optimising one objective at the expense of the others. This is also caused by the fact that the objectives are in different units and of different scales (e.g., the electricity cost has a larger scale than the migration cost). Collecting all the feasible solutions during the optimisation of every vector may then be a good improvement: see the graphs at the right of Figure 4. We notice that using a small optimality tolerance gap gives us better results than using a larger one. We also notice that we get an improvement in terms of hypervolume. This is an interesting behaviour especially since we did not add any noticeable extra computation time (CPLEX already collects the intermediary solutions, and filtering/removing the dominated solutions requires a negligible execution time). In the rest of our evaluations we collect all intermediate (good, i.e., non-dominated) solutions.



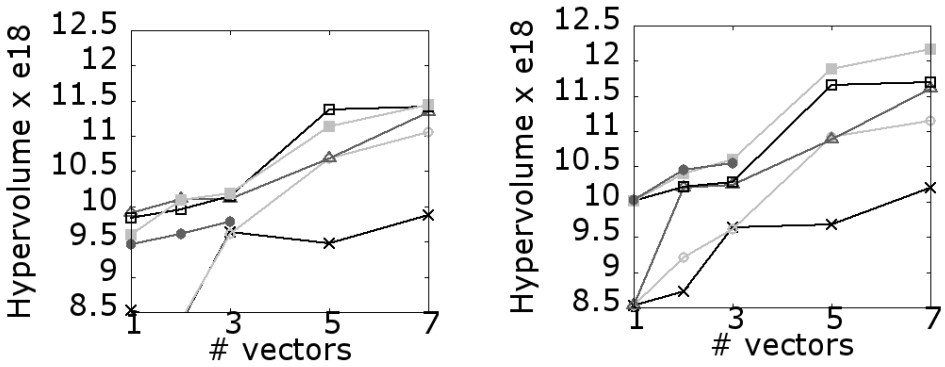
a_1_1


Hypervolume x e18
6.8
6.6
6.4
6.2
6
5.8
5.6
5.4
1
3
5
7
vectors

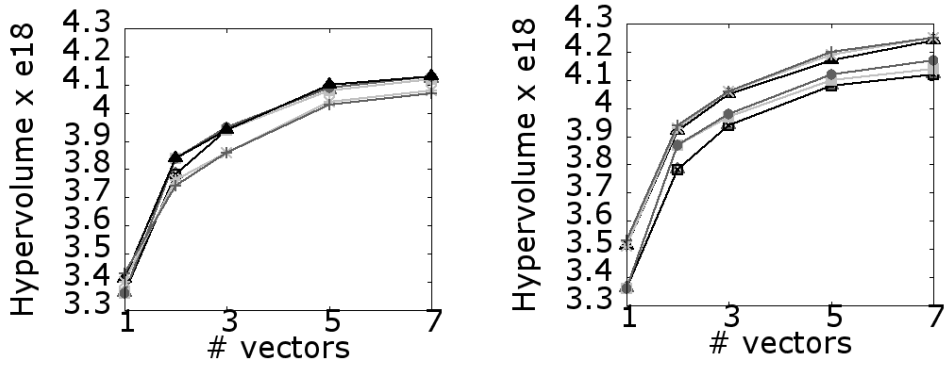
a_1_2


Hypervolume x e18
9.8
9.6
9.4
9.2
9
8.8
8.6
8.4
8.2
1
3
5
7
vectors

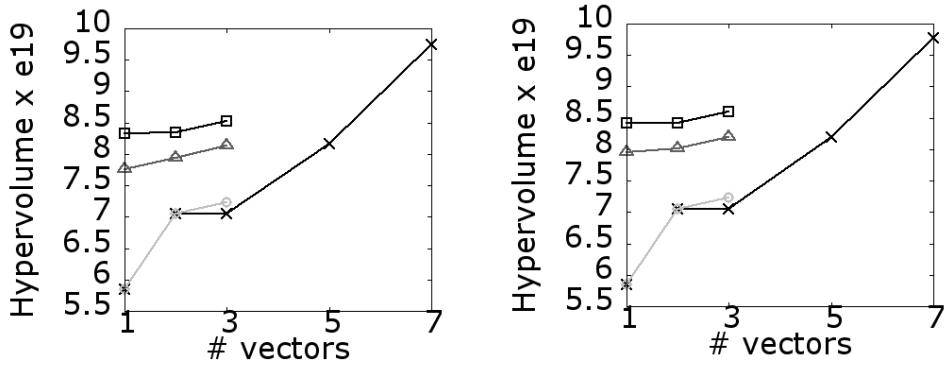
a_1_3


Hypervolume x e18
12.5
12
11.5
11
10.5
10
9.5
9
8.5
1
3
5
7
vectors

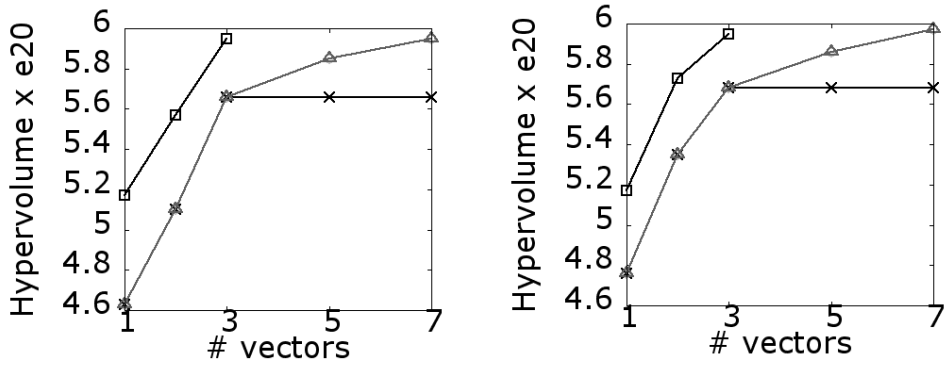
a_1_4


Hypervolume x e18
vectors
Hypervolume x e18
vectors

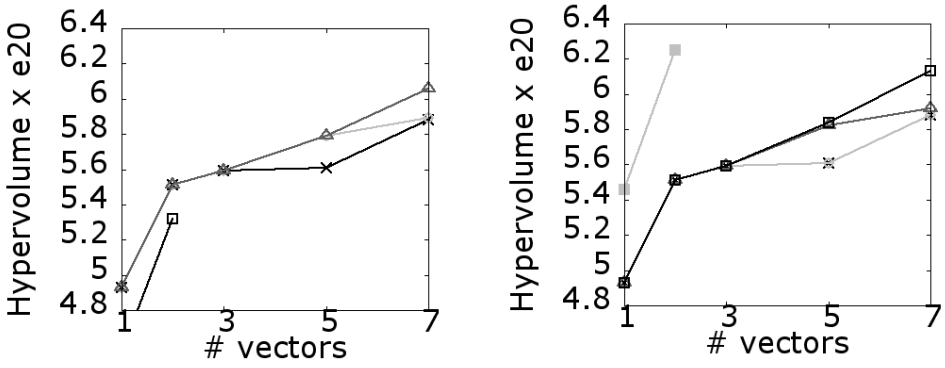
a_1_5


Hypervolume x e19
vectors
Hypervolume x e19
vectors

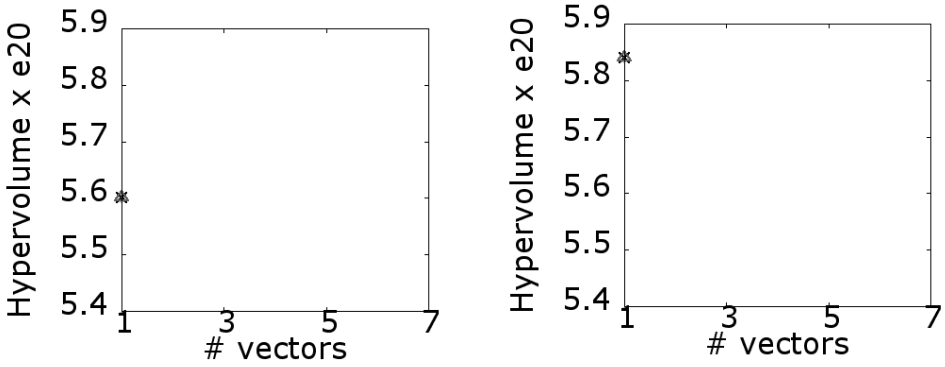
a_2_1


Hypervolume x e20
vectors
Hypervolume x e20
vectors

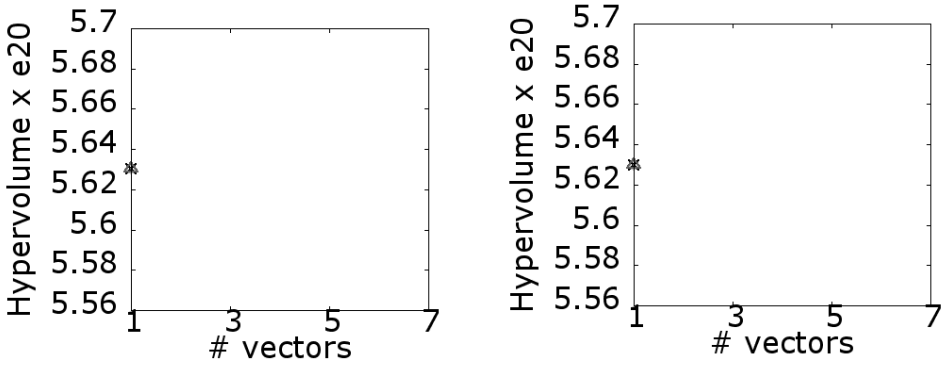
a_2_2


Hypervolume x e20
6.4
6.2
6
5.8
5.6
5.4
5.2
5
4.8
1
3
5
7
vectors

a_2_3


Hypervolume x e20
5.9
5.8
5.7
5.6
5.5
5.4
1
3
5
7
vectors

a_2_4


Hypervolume x e20
5.7
5.68
5.66
5.64
5.62
5.6
5.58
5.56
1
3
5
7
vectors

a_2_5

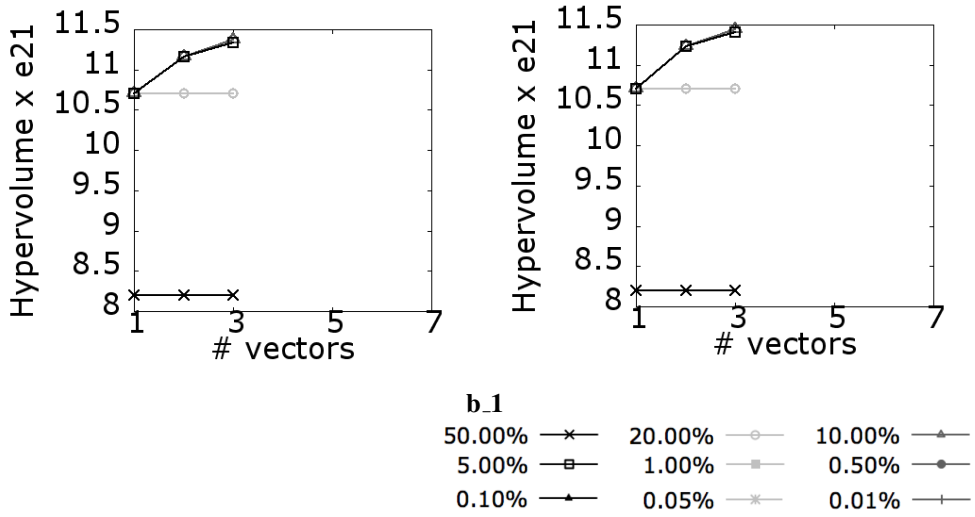


Figure 4: Hypervolume obtained on the different ROADEF instances using CPLEX for different numbers of weight vectors with different optimality gaps by either only considering the best solution found or by collecting all the feasible solutions during the optimisation.

Right most figures in Figure 4 show the hypervolumes obtained with different number of vectors and different tolerance gaps, and this can be read together with Figure 3 to figure what composition of number of vectors and tolerance gap gives the best time-hypervolume trade-off. We notice that globally the hypervolume increases with every new vector. This improvement is more noticeable during the 5 first vectors. It seems to stagnate between the 5^{th} and 7^{th} (whenever CPLEX reaches them in the time constraint), especially when the optimality gap is tiny. This leads us to think that running several CPLEX optimisations with a large number of vectors (larger than 5) would not be as beneficial as we might think. It would increase the execution time without improving the hypervolume. This idea can obviously be withdrawn if the managers of the data centre are ready to spend more time to achieve better hypervolume results.

We see from Figures 3 and right most figures in Figure 4 that using an optimality gap of 5% and 3 weight vectors for small instances and an optimality gap of 10% and 1 weight vector for medium instances allow us to achieve a good hypervolume while keeping the execution time reasonable for all the instances.

6. CBLNS for the VM Reassignment Problem

Although CPLEX has good results in terms of hypervolume, it does not deal well with large scale instances (i.e., instances of the set *b*). Therefore, it is tempting to try other matheuristic solutions which are know to work well in a limited time.

The ROADEF challenge is not multi-objective and the many algorithms proposed to

solve it cannot be applied directly to our multi-objective formulation. However, it is always possible to come up with a 'weak' version of multi-objective resolution using a mono-objective algorithm running on some weight vectors in the search space. This is for instance what we have done in the previous section with CPLEX.

Among the algorithms designed to tackle the ROADEF challenge (mono-objective VM Reassignment), Constraint-Based Large Neighbourhood Search (CBLNS) is the most promissing. CBLNS finds good solutions in a reasonable time for the mono-objective VM Reassignment Problem even on large instances.

6.1. *Description of CBLNS*

As its name indicates it, CBLNS uses a Large Neighbourhood Search (LNS [10]) metaheuristic for optimising the mono-objective VM reassignment problem.

Figure 5 gives a visual representation of the LNS algorithm. LNS starts by considering the initial assignment as the initial solution. It then selects a subset of machines and VMs from the current assignment to be reallocated, and it creates a reassignment subproblem with only those selected machines and VMs. This subproblem is solved using some optimisation techniques. If the solution to the subproblem decreases the overall cost of the global problem, then the current solution is updated by integrating the solution to the subproblem. These steps (i.e., creating a subproblem from a selection of machines and VMs and updating the current solution) are repeated for as long as the allowed time is not exceeded.

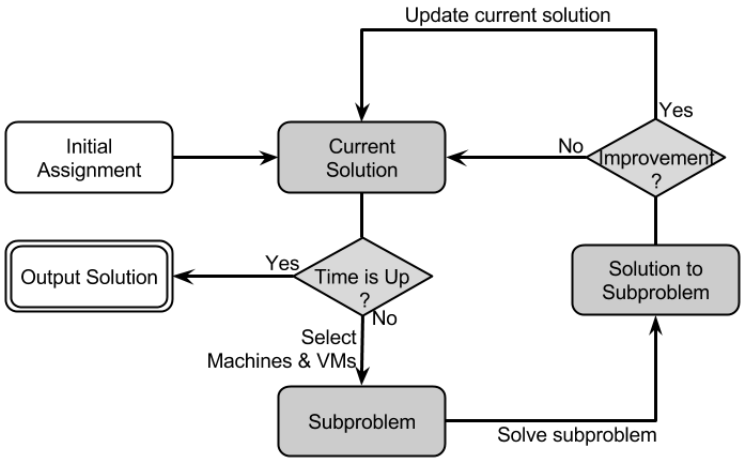


Figure 5. Flowchart representation of the Large Neighbourhood Search Algorithm.

CBLNS is a modification of LNS that proposes a novel selection of machines and VMs and resolution of the subproblems.

6.1.1. *Selection of Machines and VMs*

The selection of machines and VMs that will compose the subproblem is the first element that distinguishes CBLNS from other LNS metaheuristics. CBLNS starts by selecting a limited number of hosts $k_m \in \mathbb{N}^*$. Then CBLNS selects from each of these machines a certain number of VMs that are currently assigned to them. The total number of VMs that will be selected is also capped to $k_v^m \in \mathbb{N}^*$ per machine m. k_m is set to 1 at the beginning and is incremented at every iteration. However, it is always reset to its initial value once it exceeds 10.

Machines are sorted based on a utility function which takes the capacity into account. At every iteration, one machine is picked based on that order, whereas the rest (i.e., the other $k_m - 1$ machines) are chosen randomly.

The number of VMs k_v^m that are selected depends on the number of chosen hosts k_m. Moreover, the number of VMs picked that belong to a same machine is restricted to half of the total number of VMs on the machine. This decision aims at limiting the number of VMs that are involved in the subproblem while not being reassigned. Empirically, $k_v^m \leq 10$ for every machine m, and $\sum_{m \in \mathcal{M}_s} k_v^m \leq 40$, with $\mathcal{M}_s$ being the machines selected for the subproblem s.

6.1.2. *Solving the Subproblem*

Different techniques for modelling/optimising the subproblem have been studied [5], though CP showed better results and is considered in CBLNS. Two main operators for solving the machine reassignment problem have been developed: (i) remove_VM_From_Machine, which takes the VM from its current machine and reassigns it, and (ii) reassign_VM_To_Machine, which sets the reassignment of a given VM to a given machine. Subproblems are optimised using a systematic search, which exits though when the number of constraint failures exceeds a certain threshold.

When it comes to applying any LNS metaheuristic on large scale problems (as it is the case with the ROADEF challenge), a large number of subproblems are created and solved. Thus, this step is critical and needs to be well optimised to achieve a high overall performance. The most efficient way to create the subproblems takes advantage of the fact that all their models share the same constraints and decision variables, and that they only differ on the domain of the latter. Therefore, an original model would be saved in memory and copied every time a subproblem is needed to be solved, and the domain of its decision variables set accordingly. Despite the efficiency of this method, it requires a large amount of memory which can be unsatisfiable in some environments. Therefore, CBLNS uses a different approach: instead of having an original model stored in memory, CBLNS defines and solves a new model/problem for every subproblem.

6.1.3. *Implementation of CBLNS*

We have adapted the code provided by Mehta et al. [5]. First, we have added the parameters required to define the electricity objective that they do not consider. Then, we have removed

the resource balancing utility function that they used and was not relevant for our problem.

6.2. *CBLNS for the Multi-objective VM Reassignment Problem*

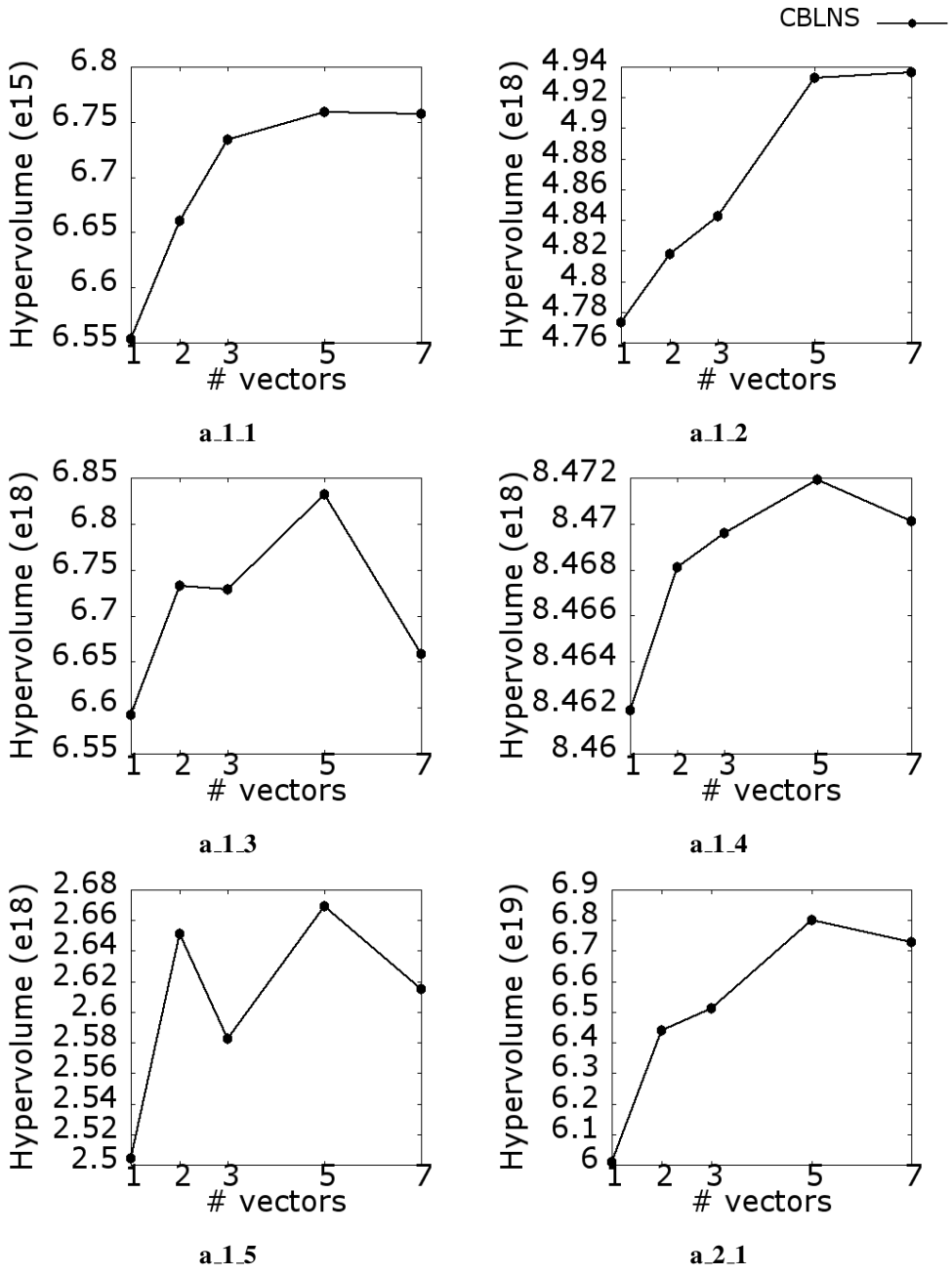
Since CBLNS is a mono-objective optimisation algorithm, we need to run it with a combination of our three considered objectives to make it suitable for our problem. As we did for CPLEX in the previous section, we combine the objectives using a weighted-sum. At every iteration, we define a vector of weights and optimise the problem using CBLNS. We consider the same vectors as CPLEX in Section 5.2. CBLNS does not consider any optimality gap: it does its optimisation as long as the execution time does not reach a predefined time limit. Thus, the more vectors we have, the less execution time we could give to each of them. For this reason, the research questions that we address here are different from the ones we evaluated before with CPLEX (see Section 5.2): what is the best trade-off (i) running CBLNS on few weight vectors over a longer period of time or (ii) running CBLNS on more weight vectors over a shorter period of time for each?

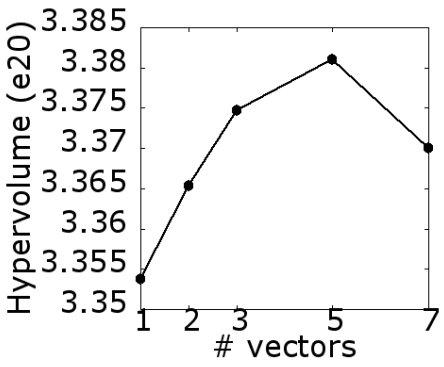
It was reported in the work of Mehta et al. [5] that the number of CBLNS' iterations goes up to 1,271,094 in only 300s, and this number is going to increase a lot knowing that our execution time is higher. Due to the large number of iterations CBLNS goes through, saving all the intermediary solutions and checking their dominance (whether they are on the Pareto front or not) would be time consuming, and degrade the quality of the optimisation. Thus, unlike what has been done for CPLEX, we only keep the best solution for each weight vector.

Figure 6 shows the hypervolume obtained using CBLNS when running with a different number of weight vectors on the different instances. We run CBLNS with every vector w for an execution time $t_i^w = \frac{T_i}{\#vectors}$, with T_i being the time limit for every instance i as shown in Table 2.

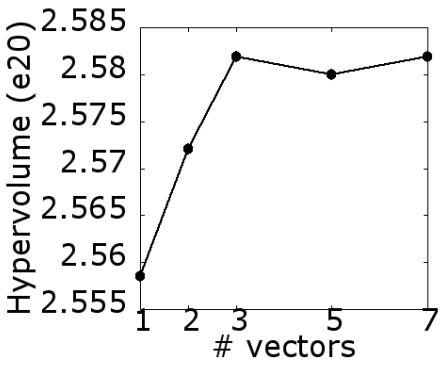
We see from Figure 6 that the hypervolume does not always increase with the number of vectors (e.g., a_1_5). This is due to the fact that the more directions are explored in the search space (i.e., the more vectors we explore) the less time we have per vector. It is also due to the fact that an improvement in the mono-objective optimisation does not automatically lead to an improvement from a multi-objective point of view. However, we notice a clear trend: the hypervolume increases till 5 vectors before decreasing in most instances. We also notice that when using 5 vectors does not achieve the best hypervolume, it either gets the second best hypervolume (i.e., a_2_5 and b_3), or a performance of nearly as good as the best number of vectors (i.e., a_1_2 and a_2_3). Based on this, using CBLNS with 5 vectors on every ROADEF instance i, with an execution time of $\frac{T_i}{5}$ per vector, seems to give the best results.

In terms of improvement, we notice that CBLNS allows getting a decent improvement, in comparison to the baseline results when only considering the initial assignment (called Initial) and the other aforementioned algorithms. And as it was expected, CBLNS also scales to large scale instances (b_2, b_3 and b_4).

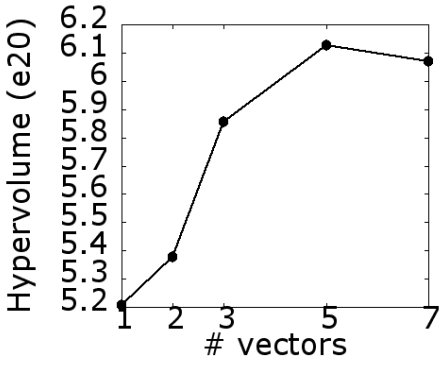

CBLNS
Hypervolume (e15)
6.8
6.75
6.7
6.65
6.6
6.55
1 2 3 5 7
vectors
a_1_1
Hypervolume (e18)
4.94
4.92
4.9
4.88
4.86
4.84
4.82
4.8
4.78
4.76
1 2 3 5 7
vectors
a_1_2
Hypervolume (e18)
6.85
6.8
6.75
6.7
6.65
6.6
6.55
1 2 3 5 7
vectors
a_1_3
Hypervolume (e18)
8.472
8.47
8.468
8.466
8.464
8.462
8.46
1 2 3 5 7
vectors
a_1_4
Hypervolume (e18)
2.68
2.66
2.64
2.62
2.6
2.58
2.56
2.54
2.52
2.5
1 2 3 5 7
vectors
a_1_5
Hypervolume (e19)
6.9
6.8
6.7
6.6
6.5
6.4
6.3
6.2
6.1
6
1 2 3 5 7
vectors
a_2_1



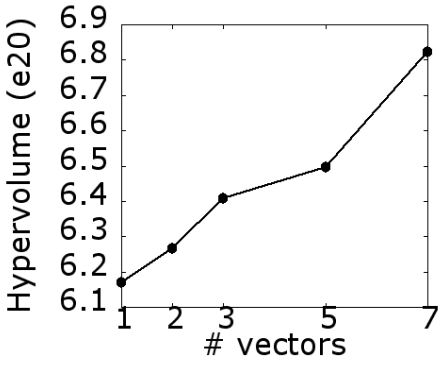
a_2_2



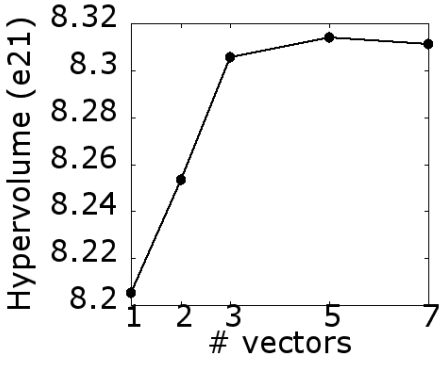
a_2_3



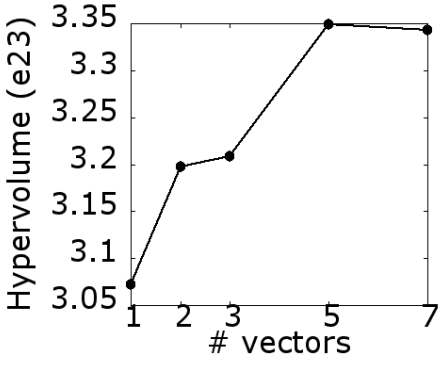
a_2_4



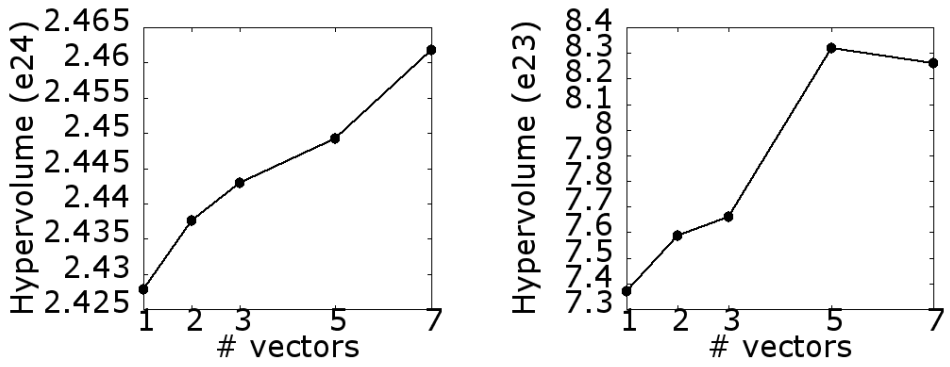
a_2_5



b_1



b_2



b_3 **b_4**

Figure 6: Hypervolume obtained with CBLNS on ROADEF instances, using maximally spread weight vectors.

7. Hybridisation

We identified in Sections 5.2 and 6.2 the most suitable parameters for CPLEX and CBLNS when addressing the multi-objective VM reassignment problem. However, it is difficult to improve the performance of these two techniques while simultaneously staying within the time limit suggested in Section 4.1. A solution to that challenge would be to combine CPLEX or CBLNS with a good heuristic algorithm that is scalable and not time consuming, which would be seeded with the good but partial first results of CPLEX or CBLNS. Therefore, instead of modifying parameters of either CPLEX or CBLNS (i.e., number of weight vectors and optimality gap), we would run them as they were previously configured in order to get a set of good solutions. This set of solutions is then given to the heuristic as a starting point for its optimisation.

A comparison between a large number of algorithms has already been performed [29] going from first-fit family techniques, metaheuristics, to hybrid-metaheuristics. A scalable hybrid-metaheuristic (i.e., GeNePi) was then proposed to optimise the multi-objective VM reassignment problem. GeNePi outperforms state-of-the-art algorithms on both quantity and quality of solutions. GeNePi successively applies three metaheuristics:

(1) GRASP (i.e., Greedy Randomized Adaptive Search Procedure [8]) to quickly find good solutions representing the search space. The greedy algorithm is run several times on our VM reassignment problem optimising the objectives weighted sum using a particular weight vector at every iteration. Weight vectors are selected on a maximally spread basis, which allows a good coverage of the search space. This step continues until finding the required number of solutions to fill out a population of a certain size.
(2) NSGA-II (i.e., Non-Sorting Genetic Algorithm [45]) combines the set of solutions from the previous step using evolutionary principles to get more and better solutions optimising different objective trade-offs. When combining two solutions together, a crossover

with a one-cut operator is used on the VM assignments per services before exchanging them, rather than applying it directly on their chromosomes (i.e., the solutions representation in the form of a chain of assignments). The crossover is followed by a mutation on the generated solutions by randomly reassigning a VM to another machine. NSGA-II uses elitism when filtering solutions between successive generations by always keeping the fittest individuals for a fast convergence.

(3) PLS (i.e., Pareto Local Search [46]) to refine the Pareto front and find more non-dominated solutions. PLS selects solutions that are the most isolated among the non-dominated ones according to the crowing metric (i.e., the distance for the solution to its two closest neighbours) and uses local search moves (i.e., a swap between two VMs assignments, and a reassignment of one VM to different machines) to explore their neighbourhood and find more solutions with similar features and characteristics.

As GeNePi is provided with a set of solutions that is resulted from either CPLEX or CBLNS, there is less necessity for applying its first step (i.e., GRASP) in order to find initial solutions representing the search space. We therefore remove the GRASP component from GeNePi and apply it only when the initial techniques fail at finding a large enough number of solutions to fill up an entire initial population and initiate the second step (i.e., NSGA-II). It might also happen that we end up with more solutions at the end of CPLEX or CBLNS than needed in NSGA-II. We therefore, filter the solutions in this case and only keep the fittest and the most spread amongst them using the crowding metric. By giving a population with good solutions representing the search space we hope that it would improve NSGA-II's performance and quicken its convergence.

We study in this section the impact of combining CPLEX or CBLNS with GeNePi both on the obtained solutions (how much qualitative and quantitative improvement do we achieve?) and on the execution time (how much is the increase in execution time?).

7.1. *Combining CPLEX and GeNePi*

We saw earlier that CPLEX gets good results on small and medium instances, but it becomes really hard to improve those results without increasing either the number of vectors or the optimality gap, and thus dramatically impacting the execution time. We therefore run CPLEX using a gap of 5% and 3 vectors for small instances, and a gap of 10% and a unique vector for medium and large instances. We also collect all the intermediary feasible solutions during CPLEX optimisation as it was shown that it is improving the quality of the obtained set of solutions without any significant execution time overhead.

We take the previous implementation of CPLEX and we give its results to GeNePi, non-dominated solutions found using CPLEX used as the initial population as shown in Figure 7. It might happen that CPLEX does not find enough solutions to fill out an entire population (in our case, a population of size 20). In this case the original greedy algorithm in GeNePi is applied to fill this gap and compensate this lack of solutions. The set of solutions from CPLEX can thus be seen as 'seeds' for GeNePi's optimisation. Similarly, the opposite can happen if CPLEX finds a set of solutions larger than the required initial population. In this case, we filter them by only keeping the fittest (the non-dominated ones)

for a faster convergence and the most isolate amongst them (using the crowding measure) for a better representativeness of the search space. In this situation, CPLEX acts more as a substitute for the first component of GeNePi (i.e., GRAPS).

In our implementation, GeNePi applies 10 generations of its second phase (i.e., NSGA-II) in order to evolve the initial population into a fitter one, by getting better and more scattered solutions (spread over the search space). At the end, GeNePi refines the Pareto front by applying a unique iteration of PLS on the 10 most isolated solutions. Although this step does not bring a large improvement in terms of hypervolume, it is important as it provides decision makers with more implementation choices. Beside these choices, we use the same parameters as in the GeNePi paper [29].

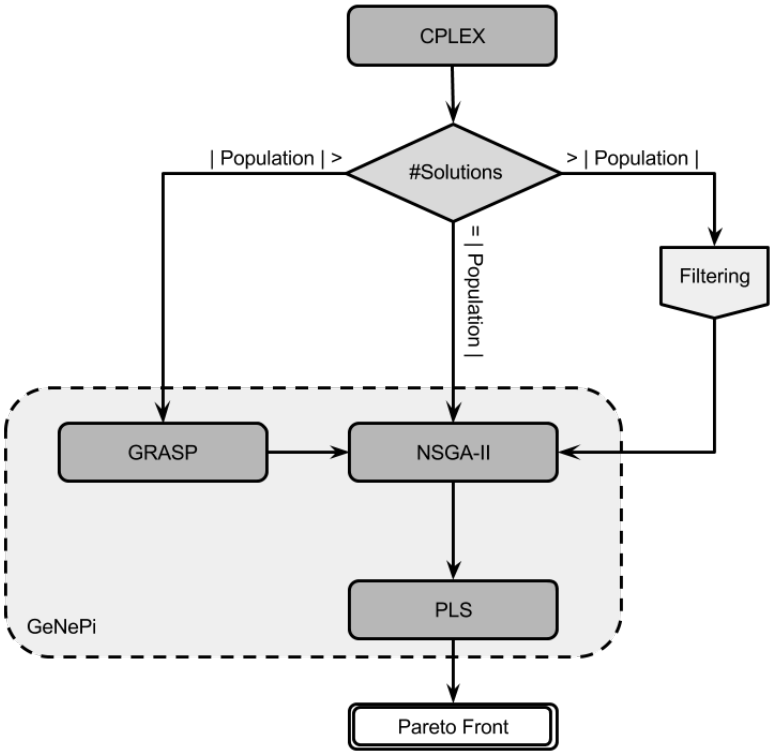


Figure 7. Flowchart representation of the combination of CPLEX and GeNePi (i.e., CPLEX+GeNePi).

7.2. *Combining CBLNS and GeNePi*

Section 6.2 showed us that CBLNS achieves good hypervolume values on both small scale (i.e., 'a' instances) and large scale instances (i.e., 'b' instances). We also saw that choosing 5 vectors seems to be the best option, and increasing this number also means decreasing

the time allowed to CBLNS on each of them, which often leads to a decrease in the hypervolume. Given that 'performance wall' we decided to mix CBLNS on the 5 vectors and GeNePi with the same parameters described in the previous section.

As it can be seen in Figure 8, we combine CBLNS with GeNePi in a similar way as how it is done with CPLEX. However, there is a difference when it comes to fitting the solutions from CBLNS to the size of the initial population. In our CBLNS, we only collect one solution (i.e., the final one) after each run of CBLNS with a particular weight vector. Given that we configure CBLNS to run with a number of vectors always smaller than the population size, we never exceed the population size. Therefore, it is never needed to filter CBLNS' resulted set of solutions. Instead, it is needed to add other solutions to fill out the initial population using the GRASP component in GeNePi. In this case, GRASP is always enabled in GeNePi and CBLNS results acts always as 'seeds' to GeNePi's optimisation.

Similarly to the implementation of GeNePi when combining it with CPLEX, we also use a population size of 20 individuals. We also provide the generated set of solutions from CBLNS and GRASP to the second phase of GeNePi (i.e., NSGA-II) as an initial population to evolve and optimise it in 10 generations. The non-dominated solutions that are found after running NSGA-II are then given to the third and last step of GeNePi (i.e., PLS). Only 10 of the most isolated solutions amongst them are selected in order to search their neighbourhoods for more solutions with similar characteristics and refine the Pareto front.

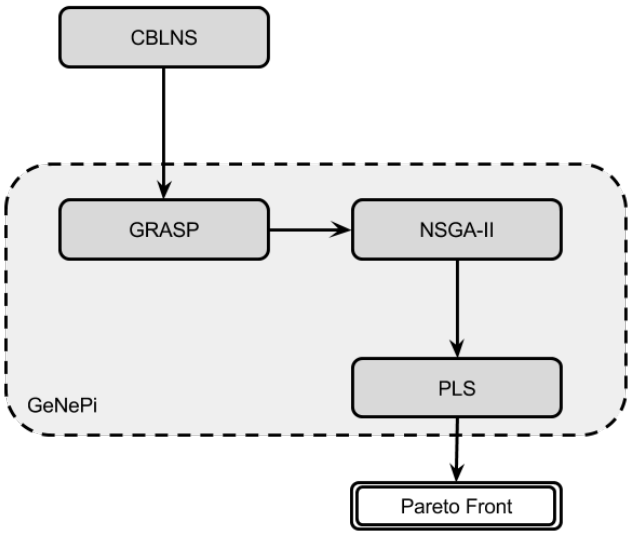


Figure 8. Flowchart representation of the combination of CBLNS and GeNePi (i.e., CBLNS+GeNePi).

Table 3. Summary of results obtained with GeNePi, CPLEX, CPLEX combined with GeNePi using a gap of 5% and 3 vectors for small instances, and a gap of 10% and a unique vector for medium and large instances, CBLNS using 5 vectors, and CBLNS combined with GeNePi, against the initial assignment.

		Initial	GeNePi	CPLEX	CPLEX+ GeNePi	CBLNS	CBLNS+ GeNePi
a_1_1	**Hyp (e15)**	6.16	9.71	8.82	9.74	6.76	9.16
	# Sol	1	225	3	30	4	124
	Time	0	2	0.18	0.38	30	31
a_1_2	**Hyp (e18)**	4.75	5.69	5.93	6.17	4.93	5.51
	# Sol	1	160	8	42	4	193
	Time	0	518	303	319	3,600	4,605
a_1_3	**Hyp (e18)**	6.57	7.22	9.01	9.06	6.83	7.24
	# Sol	1	132	5	21	6	129
	Time	0	441	151	167	3,600	4,375
a_1_4	**Hyp (e18)**	8.10	9.19	10.28	10.89	8.47	8.96
	# Sol	1	136	6	35	3	200
	Time	0	309	138	158	3,600	4,184
a_1_5	**Hyp (e18)**	2.42	3.15	3.94	4.04	2.67	3.11
	# Sol	1	283	4	54	4	266
	Time	0	332	21	34	3,600	4,271
a_2_1	**Hyp (e19)**	4.57	5.89	7.95	8.17	6.80	7.26
	# Sol	1	231	5	39	6	380
	Time	0	284	159	166	7,200	8,786
a_2_2	**Hyp (e20)**	3.33	4.36	4.76	5.01	3.38	4.37
	# Sol	1	197	7	42	3	195
	Time	0	600	2,580	2,694	7,200	8,315
a_2_3	**Hyp (e20)**	2.55	3.69	4.93	5.13	2.58	3.73
	# Sol	1	202	2	32	2	242
	Time	0	695	71	154	7,200	8,471
a_2_4	**Hyp (e20)**	3.21	6.19	5.84	6.68	6.13	9.79
	# Sol	1	253	3	43	3	429
	Time	0	342	20,655	21,173	36,000	36,558
a_2_5	**Hyp (e20)**	4.96	5.81	5.63	5.86	6.50	7.30
	# Sol	1	220	2	23	5	414
	Time	0	347	22,513	22,548	36,000	36,591
b_1	**Hyp (e21)**	8.20	8.74	10.7	10.88	8.31	8.53
	# Sol	1	244	2	29	5	30
	Time	0	14,991	8,913	9,452	36,000	43,225
b_2	**Hyp (e23)**	1.43	3.29			3.35	3.36
	# Sol	1	300	-	-	6	461
	Time	0	10,028			36,000	44,249
b_3	**Hyp (e24)**	2.33	2.41			2.45	3.76
	# Sol	1	162	-	-	3	30
	Time	0	39,596			36,000	43,221
b_4	**Hyp (e23)**	3.65	4.21			8.32	8.34
	# Sol	1	118	-	-	6	41
	Time	0	63,535			36,000	43,392

Note: The symbol '–' indicates that the algorithm could not run on the corresponding instance.

7.3. *Comparative Study*

Table 3 shows the results obtained on the modified ROADEF instances from a_1_1 to b_4 in terms of hypervolume, number of non-dominated solutions and execution time. Results

are obtained using 10 runs of GeNePi alone (the average is taken), CPLEX alone, CBLNS alone, and our two new matheuristics: (i) CPLEX combined with GeNePi while respectively defining the optimality gap and the number of weight vectors to 5% and 3 for small instances (i.e., a_1_x), and 10% and 1 for medium and large ones (i.e., a_2_x and b_1), and (ii) CBLNS combined with GeNePi after running CBLNS with 5 weight vectors.

Table 3 confirms that GeNePi succeeds in improving the hypervolume and getting a large number of non-dominated solutions while keeping the execution time relatively low.

We also notice that CPLEX based techniques (i.e., CPLEX and CPLEX+GeNePi) can not optimise large scale instances (i.e., b_2, b_3 and b_4). However, CPLEX outperforms GeNePi in terms of hypervolume on 8 instances out of the 11 other instances (i.e., a_x and b_1) with an average improvement of 102%. But, GeNePi gets on average 63 times more non-dominated solutions (note that the number of solutions, while interesting for the decision makers, is not as important as the hypervolume, and anyway too many solutions makes them difficult to explore). We see that CPLEX is slightly better in terms of execution time (given its parallel implementation and the fact that it runs on more powerful memory intensive machines). However we also see that CPLEX struggles to scale to large instances (GeNePi gets a better hypervolume on a_2_4 and a_2_5 with an execution time of respectively 342s and 347s vs. 21,173s and 22,548s for CPLEX). Compared to CPLEX, we clearly see that adding GeNePi to CPLEX helps to improve the hypervolume (an increase of 17.84% on average), and also to get more non-dominated solutions (8.9 times more solutions on average), while keeping the execution time low (an average execution time increase of 31.10%, but of only 6% for execution times larger than 100s). We also see that CPLEX+GeNePi outperforms the hypervolume obtained by GeNePi alone (with an average increase of 126.96%) and that unlike CPLEX alone, CPLEX+GeNePi gets better hypervolumes than GeNePi in all instances. CPLEX+GeNePi also gets a fairly reasonable number of non-dominated solutions. However, GeNePi still gets a larger number of solutions (5 times more on average).

On the other hand, we see that CBLNS alone only gets a small number of non-dominated solutions (GeNePi finds 52 times more non-dominated solutions on average). This number stays within the range of CPLEX' results, though. Moreover, CBLNS alone does not perform as well as GeNePi and CPLEX in terms of hypervolume on small and medium instances with the exception of a_2_1 and a_2_5, though, unlike CPLEX, it is able to optimise large scale instances. It is even outperforming GeNePi in terms of hypervolume on most of the large instances (i.e., b_2, b_3 and b_4). On average, despite CBLNS' small number of solutions, it achieves an improvement of more than 22.15% in hypervolume in comparison to GeNePi, but it is only reaching 45.2% of the hypervolume achieved by CPLEX on average on the instances it could optimise, with an execution time nearly 20% larger. In comparison to CBLNS alone, CBLNS+GeNePi gets way more interesting results. CBLNS+GeNePi gets 56.76 times more number of non-dominated solutions with an hypervolume 6 times greater than those of CBLNS, while only requiring 16.4% extra execution time. This improvement makes CBLNS+GeNePi better in hypervolume than GeNePi on 11 instances out of 14, which represents more than 2 times improvement in hypervolume and nearly 57 times increase in non-dominated solutions on average. Compared

to CPLEX+GeNePi, we notice a same behaviour as what was seen between CBLNS and CPLEX: CPLEX+GeNePi is better on small instances, while CBLNS+GeNePi gets more interesting when the scale of the instances increases (i.e., from a_2_4 to b_4, b_1 excluded).

To summarise, we can say that CPLEX is good at getting few solutions with a good quality but does not scale well to large instances. GeNePi gets a relatively good performance overall but is often outperformed in hypervolume by either CPLEX or CBLNS (except on a_1_1 and a_2_4). CBLNS does not perform as well as GeNePi and CPLEX on small and medium instances, but outperforms both of them when it comes to large scale instances. Also, knowing that CPLEX' execution time on one vector is low does not provide much information neither on the time to run on several other vectors, nor on their performance. Whereas CBLNS' execution time is given as an input and is divided between every weight vector, and the increase in the number of vectors often leads to a decrease in performance after a certain threshold (i.e., 5 in our case).

Combining CPLEX with GeNePi seems to be a good solution to improve CPLEX' results in both hypervolume and number of non-dominated solutions with a relatively low increase in the execution time. We have a similar concluding remark for CBLNS with GeNePi: the hybridisation brings better results with a limited extra time required. The advantage of our approach is that it can be adapted to the size of the problem: when the size increases, we have the option to either continue relaxing some of the parameters used for CPLEX (i.e., less vectors or a larger optimality gap) or to replace CPLEX with CBLNS to feed in GeNePi.

8. Conclusion

This paper surveys how classical exact and hybrid optimisation techniques, such as MILP, CBLNS and the likes, perform on the Multi-objective VM Reassignment Problem, a large and difficult problem with a lot of complex constraints: given some reassignment objectives (e.g., electricity cost, migration cost, reliability cost), find the best set of reassignment solutions in a limited time – the limit being quite large (10 hours for large instances). We show that a MILP solver (CPLEX) can be used with some relaxations: allowing an optimality tolerance gap (which stops CPLEX when the solutions found are close to the optimal) and limiting the number of directions explored in the search space (giving CPLEX only few vectors of weighted objectives to explore). We also show limitations of CPLEX when dealing with very large scale instances, and that CBLNS, while not performing as well as CPLEX on small and medium instances, scales seamlessly to larger instances.

We also propose to feed the results of CPLEX and CBLNS to a metaheuristic (GeNePi) and we compare CPLEX alone, CBLNS alone, GeNePi alone, CPLEX+GeNePi and CBLNS+GeNePi. We observe that CPLEX is better than both GeNePi (an improvement of the hypervolume of 102% in comparison to GeNePi on average) and CBLNS (CBLNS only achieves 45% of the hypervolume obtained using CPLEX on average). However, CPLEX is limited to small and medium instances (a_x and b_1 in our benchmark). On the other hand, GeNePi and CBLNS scale to large instances – CBLNS gets an improvement of 22% in hypervolume in comparison to GeNePi, despite being dominated on most of small and

medium instances (GeNePi outperforms CBLNS in hypervolume on 9 instances out of 14). CPLEX+GeNePi outperforms both CPLEX and GeNePi in terms of hypervolume (an average increase of 126.9% vs. GeNePi and 17.8% vs. CPLEX), while the execution time remains acceptable (an increase of only 6% on average in comparison to CPLEX for execution times larger than 100s). CBLNS+GeNePi outperforms both CBLNS and GeNePi in terms of hypervolume (an average increase of 201.67% vs. GeNePi and 615.13% vs. CBLNS), while the execution time remains acceptable (with an increase of 16.40% on average in comparison to CBLNS). Overall, CPLEX+GeNePi outperforms CBLNS+GeNePi on small and medium instances, but CBLNS+GeNePi can scale to large instances.

Acknowledgement

This work was supported, in part, by Science Foundation Ireland grants 10/CE/I1855 and 13/RC/2094, and by Science Foundation Ireland Industry Fellowship grant 13/IF/12789.

Bibliography

1. A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *FGCS*, pp. 755–768, 2012.
2. R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *JNCA*, pp. 11–25, 2015.
3. A. Corradi, M. Fanelli, and L. Foschini, "Vm consolidation: A real case based on openstack cloud," *FGCS*, pp. 118–127, 2014.
4. F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in virtualized datacenters," in *CP*, pp. 27–41, 2011.
5. D. Mehta, B. O'Sullivan, and H. Simonis, "Comparing solution methods for the machine reassignment problem," in *CP*, pp. 782–797, 2011.
6. H. Gavranović, M. Buljubašić, and E. Demirović, "Variable neighborhood search for google machine reassignment problem," *ENDM*, pp. 209–216, 2012.
7. F. Brandt, J. Speck, and M. Völker, "Constraint-based large neighborhood search for machine reassignment," *Ann Oper Res*, pp. 1–29, 2012.
8. M. Gabay and S. Zaourar, "A GRASP approach for the machine reassignment problem," in *EURO*, 2012.
9. "Google/roadef challenge'12." `http://challenge.roadef.org/2012/en/`.
10. P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *CP*, pp. 417–431, 1998.
11. R. L. Graham, "Bounds on multiprocessing anomalies and related packing algorithms," in *Proceedings of the May 16-18, 1972, spring joint computer conference*, pp. 205–217, 1972.
12. A. Caprara and P. Toth, "Lower bounds and algorithms for the 2-dimensional vector packing problem," *Discrete Applied Mathematics*, pp. 231–262, 2001.
13. H. Kellerer and V. Kotov, "An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing," *Operations Research Letters*, pp. 35–41, 2003.
14. J. Beck and D. Siewiorek, "Modeling multicomputer task allocation as a vector packing problem," in *Proceedings of the 9th international symposium on System synthesis*, p. 115, 1996.
15. W. Leinberger, G. Karypis, and V. Kumar, "Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints," in *ICPP*, pp. 404–412, 1999.
16. K. Jansen and S. Öhring, "Approximation algorithms for time constrained scheduling," *Information and Computation*, pp. 85–108, 1997.

17. M. Gendreau, G. Laporte, and F. Semet, "Heuristics and lower bounds for the bin packing problem with conflicts," *Computers & Operations Research*, pp. 347–358, 2004.
18. R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.
19. H. Shachnai and T. Tamir, "Approximation schemes for generalized two-dimensional vector packing with application to data placement," *Journal of Discrete Algorithms*, pp. 35–48, 2012.
20. S. K. Doddavula, M. Kaushik, and A. Jain, "Implementation of a fast vector packing algorithm and its application for server consolidation," in *CloudCom)*, pp. 332–339, 2011.
21. M. Stillwell, F. Vivien, and H. Casanova, "Virtual machine resource allocation for service hosting on heterogeneous distributed platforms," in *IPDPS*, pp. 786–797, 2012.
22. 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 Systems with Applications*, pp. 5266–5275, 2013.
23. G. M. Portal, M. Ritt, L. M. Borba, and L. S. Buriol, "Simulated annealing for the machine reassignment problem," *AOR*, pp. 1–22, 2012.
24. W. Jaśkowski, M. Szubert, and P. Gawron, "A hybrid mip-based large neighborhood search heuristic for solving the machine reassignment problem," *Ann Oper Res*, pp. 1–30, 2015.
25. F. Lopez Pires and B. Baran, "Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach," in *UCC*, pp. 203–210, 2013.
26. A. Sallam and K. Li, "A multi-objective virtual machine migration policy in cloud systems," *The Computer Journal*, pp. 195–204, 2014.
27. G. Jung, M. Hiltunen, K. R. Joshi, R. D. Schlichting, C. Pu, *et al.*, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *ICDCS*, pp. 62–73, 2010.
28. J. Xu and J. Fortes, "A multi-objective approach to virtual machine management in datacenters," in *ICAC*, pp. 225–234, 2011.
29. T. Saber, A. Ventresque, X. Gandibleux, and L. Murphy, "Genepi: A multi-objective machine reassignment algorithm for data centres," in *HM*, pp. 115–129, 2014.
30. T. Saber, A. Ventresque, J. Marques-Silva, J. Thorburn, and L. Murphy, "Milp for the multi-objective vm reassignment problem," in *ICTAI*, pp. 41–48, 2015.
31. T. Saber, A. Ventresque, L. Murphy, and E.-G. Talbi, "Multi-objective vm reassignment for the enterprise," in *META*, 2014.
32. T. Saber, A. Ventresque, I. Brandic, J. Thorburn, and L. Murphy, "Towards a multi-objective vm reassignment for large decentralised data centres," in *UCC*, 2015.
33. 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*, Intel Technology Journal, 2008.
34. W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *CloudCom*, pp. 254–265, 2009.
35. B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *TDSC*, pp. 337–351, 2010.
36. X. Li, A. Ventresque, J. Omana Iglesias, and J. Murphy, "Scalable correlation-aware virtual machine consolidation using two-phase clustering," in *HPCS*, 2015.
37. J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *GreenCom*, pp. 179–188, 2010.
38. C.-H. Lien, Y.-W. Bai, and M.-B. Lin, "Estimation by software for the power consumption of streaming-media servers," *TIM*, pp. 1859–1870, 2007.
39. H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Computing*, pp. 249–264, 2013.
40. V. De Maio, G. Kecskemeti, and R. Prodan, "A workload-aware energy model for virtual ma-

chine migration," in *IEEE Cluster*, pp. 274–283, 2015.
41. K. Rybina, W. Dargie, A. Strunk, and A. Schill, "Investigation into the energy cost of live migration of virtual machines," in *SustainIT*, pp. 1–8, 2013.
42. E. Zitzler, M. Laumanns, L. Thiele, C. M. Fonseca, and V. G. da Fonseca, "Why quality assessment of multiobjective optimizers is difficult.," in *GECCO*, pp. 666–673, 2002.
43. E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms. a comparative case study," in *PPSN*, pp. 292–301, 1998.
44. K. Bringmann and T. Friedrich, "Approximation quality of the hypervolume indicator," *Artificial Intelligence*, pp. 265–290, 2013.
45. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *TEVC*, pp. 182–197, 2002.
46. A. Alsheddy and E. E. Tsang, "Guided pareto local search based frameworks for biobjective optimization," in *CEC*, 2010.