<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Multi-objective Virtual Machine Reassignment for Large Data Centres</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authors(s)</strong></td>
<td>Saber, Takfarinas</td>
</tr>
<tr>
<td><strong>Publication date</strong></td>
<td>2017</td>
</tr>
<tr>
<td><strong>Publisher</strong></td>
<td>University College Dublin. School of Computer Science</td>
</tr>
<tr>
<td><strong>Link to online version</strong></td>
<td><a href="http://dissertations.umi.com/ucd:10168">http://dissertations.umi.com/ucd:10168</a></td>
</tr>
<tr>
<td><strong>Item record/more information</strong></td>
<td><a href="http://hdl.handle.net/10197/9532">http://hdl.handle.net/10197/9532</a></td>
</tr>
</tbody>
</table>

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)
MULTI-OBJECTIVE
VIRTUAL MACHINE REASSIGNMENT
 FOR LARGE DATA CENTRES

by

Takfarinas Saber
Student ID: 12252605

The thesis is submitted to University College Dublin in fulfilment of the
requirements for the degree of
Doctor of Philosophy

School of Computer Science
University College Dublin

Head of School:
Prof. Pádraig Cunningham

Under the supervision of:
Prof. Liam Murphy & Dr Anthony Ventresque

Doctoral Study Panel:
Prof. João Marques-Silva
Prof. John Murphy
Prof. Liam Murphy

July 2017
CONTENTS

Abstract iv
Declaration v
Acknowledgements vi
List of Figures viii
List of Tables x
List of Acronyms xi
Publications xiii

1 Introduction 1
  1.1 Motivation .............................................. 2
  1.2 Problem Definition ..................................... 4
  1.3 Thesis Approach ....................................... 5
  1.4 Contributions ......................................... 7
  1.5 Thesis Structure ...................................... 8

2 Background and Related Work 10
  2.1 Background .............................................. 10
    2.1.1 Virtualisation ..................................... 10
    2.1.2 Consolidation Through VM Placement ............... 12
    2.1.3 Public Clouds ..................................... 17
    2.1.4 Resolution vs. Optimisation ....................... 18
    2.1.5 Evaluation of Multi-objective Optimisation Techniques . 24
  2.2 Related Work .......................................... 28
    2.2.1 VM Reassignment in Centralised Data Centres ...... 28
    2.2.2 VM Reassignment in Distributed Data Centres ...... 37
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3</td>
<td>Multi-objective VM Reassignment</td>
<td>49</td>
</tr>
<tr>
<td>2.3</td>
<td>Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>3</td>
<td>Exact and Hybrid Algorithms for the Multi-objective VM Reassignment</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Problem Definition</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>3.2.1 Problem Description and Notation</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>3.2.2 Constraints</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>3.2.3 Objectives</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>3.3 Experimental Setup</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>3.3.1 Dataset</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>3.3.2 Metrics</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>3.4 CPLEX for the VM Reassignment Problem</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>3.4.1 CPLEX for the Mono-objective VM Reassignment Problem</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>3.4.2 CPLEX for the Multi-objective VM Reassignment Problem</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>3.5 CBLNS for the VM Reassignment Problem</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>3.5.1 Description of CBLNS</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>3.5.2 CBLNS for the Multi-objective VM Reassignment Problem</td>
<td>97</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusion</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>Heuristics, Metaheuristics and Hybrid Techniques for the Multi-objective VM Reassignment Problem</td>
<td>102</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>102</td>
</tr>
<tr>
<td>4.2</td>
<td>Description of the Solution: GeNePi</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>4.2.1 Ge: a Variant of the Constructive Phase of GRASP</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>4.2.2 Ne: NSGA-II</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>4.2.3 Pi: a Pareto Local Search</td>
<td>106</td>
</tr>
<tr>
<td>4.3</td>
<td>Experimental Setups</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>4.3.1 Algorithms</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>4.3.2 Tuning the Steps of GeNePi</td>
<td>108</td>
</tr>
<tr>
<td>4.4</td>
<td>Evaluation of GeNePi Against (Meta)Heuristics</td>
<td>110</td>
</tr>
<tr>
<td>4.5</td>
<td>Hybridisation</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>4.5.1 Combining CPLEX and GeNePi</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>4.5.2 Combining CBLNS and GeNePi</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>4.5.3 Comparative Study</td>
<td>123</td>
</tr>
<tr>
<td>4.6</td>
<td>Evaluation Against an Exact Resolution</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>4.6.1 Description of the Epsilon Constraints Method</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>4.6.2 Implementation of the Epsilon Constraints Method</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>4.6.3 Results Obtained Using Epsilon Constraints Method</td>
<td>129</td>
</tr>
<tr>
<td>4.7</td>
<td>Conclusion</td>
<td>134</td>
</tr>
</tbody>
</table>
## CONTENTS

5 The Decentralised Data Centre Context 136
  5.1 Introduction ................................. 136
  5.2 Formal Problem Definition .................. 138
    5.2.1 Constraints of the Problem .............. 139
    5.2.2 Objectives to Optimise .................. 140
  5.3 E-GeNePi ................................. 141
    5.3.1 Reassignment ......................... 142
    5.3.2 Placement ....................... 143
  5.4 Experimental Setup ....................... 145
    5.4.1 Other Algorithms ....................... 147
  5.5 Evaluation ............................... 147
    5.5.1 Evaluation of the VCs’ Hill Climbing .... 148
    5.5.2 Quantity and Quality of Solutions ....... 151
    5.5.3 Profiling ............................ 153
  5.6 Conclusion ............................... 156

6 The Hybrid Cloud Context 157
  6.1 Introduction .............................. 157
  6.2 Formal Problem Definition .................. 159
    6.2.1 Constraints of the Problem .............. 160
    6.2.2 Objectives to Optimise .................. 162
  6.3 H2-D2 ..................................... 163
    6.3.1 Reassignment ......................... 165
  6.4 Experimental Setup ....................... 166
    6.4.1 Other Algorithms ....................... 169
  6.5 Evaluation ................................ 171
    6.5.1 Quantitative and Qualitative Evaluation .... 171
    6.5.2 Profiling ............................ 176
  6.6 Conclusion ............................... 181

7 CONCLUSION 182
  7.1 Summary and Contributions of the Thesis .... 182
  7.2 Possible Directions for Future Work .......... 183

References 186
Abstract

Data centres are large IT facilities composed of an intricate collection of interconnected and virtualised computers, connected services, and complex service-level agreements. Optimising data centres, often attempted by reassigning virtual machines to servers, is both desirable and challenging. It is desirable as it could save a large amount of money: using servers better would lead to decommissioning unused ones and organising services better would increase reliability and maintenance. It is also challenging as the search space is very large and very constrained, which makes the solutions difficult to find. Moreover, in practice assignments can be evaluated from different perspectives, such as electricity cost, overall reliability, migration overhead and cloud cost. Managers in data centres then make complex decisions and need to manipulate possible solutions favouring different objectives to find the right balance. Another element I consider in the context of this work is that organisations hosting large IT facilities are often geographically distributed – which means these organisations are composed of a number of hosting departments which have different preferences on what to host and where to host it, and a certain degree of autonomy. The problem is even more challenging as companies can now choose from a pool of public cloud services to host some of their virtual machines.

In this thesis, I address the problem of multi-objective virtual machine (VM) reassignment for large data centres from three realistic and challenging perspectives.

- First, I demonstrate how intractable is the exact resolution of the problem in a centralised context: I perform a thorough performance evaluation of classical solvers and metaheuristics, and I propose a novel hybrid algorithm which outperforms them.
- Second, I design a two-level system addressing multi-objective VM reassignment for large decentralised data centres. My system takes care of both the reassignment of VMs and their placement within the hosting departments and I propose algorithms that optimise each of the levels.
- Third, I extend my work to the hybrid cloud world – i.e., when companies can decide to use their own internal resources or pay for public clouds computing resources. The problem becomes now more dynamic (as prices evolve) and challenging, and I propose a novel algorithm that takes all these elements into account.
I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the Title Page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

Signed:   

Student ID:   12252605

Date:   28/07/2017
Acknowledgements

I am grateful to the Irish Software Research Centre (Lero) for providing the funding that made this work possible. I am also grateful to the Performance Engineering Laboratory (PEL) in University College Dublin (UCD) for hosting and providing me with the resources needed to conduct this work.

I would like to thank my supervisors Prof. Liam Murphy and Dr Anthony Ventresque for giving me the opportunity to do my PhD and for their valuable guidance and great support throughout this journey.

I would also like to express my gratitude to my collaborators: Prof. Xavier Gandibleux, Prof. Joao Marques-Silva, Prof. Ivona Brandic, Prof. El-Ghazali Talbi, Dr Yue Jin and Mr James Thorburn for sharing with me their vast knowledge and research expertise in their respective fields.

A very special gratitude goes out to my colleagues in PEL for their support and for all the nice time we spent together during all these years.

I am in debt to many of my friends, especially Quentin and Malika for putting up with me and managing to cheer me up all these years.

I am grateful to my parents Hocine and Djouher, my brothers Merouane, Amar and Mouloud, and my sisters Dounia and Ouarda for their unconditional love and unwavering support.

_Dublin, May 2017_  
_Takfarinas Saber_
LIST OF FIGURES

1.1 Thesis approach: from a ‘single’ centralised data centre, to a hybrid data centre, via a decentralised data centre. .......... 6

2.1 Comparison of hypervisor- and container-based virtualisation technologies. .......................................................... 13

2.2 Distribution of teams taking part in the Google ROADEF/EURO 2012 Challenge. ..................................................... 32

3.1 Overview of centralised data centres. ................................. 72

3.2 Simple scenario of a correct assignment of VMs to PMs. ...... 77

3.3 Metrics: number of non-dominated solutions and hypervolume. 83

3.4 Execution time of CPLEX on the Google ROADEF/EURO challenge instances. .............................. 88

3.5 Hypervolume obtained on the different Google ROADEF/EURO 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. 93

3.6 Flowchart representation of the Large Neighbourhood Search Algorithm. ....................................................... 95

3.7 Average hypervolume of 10 runs obtained with CBLNS on the Google ROADEF/EURO instances, using maximally spread weight vectors. ............................................................... 100

4.1 Hypervolume obtained with Ge using different values for the parameter $\alpha$. ....................................................... 108

4.2 Hypervolume obtained with GeNe by setting $\alpha$ to 0.6, size of the population to 50 and the number of iterations to 100, while varying both $P_c$ and $P_m$. ................................. 109
4.3 Average improvement curves of 10 runs of the different algorithms on the different instances. .......................... 118
4.4 Flowchart representation of CPLEX+GeNePi. .................. 121
4.5 Flowchart representation of CBLNS+GeNePi. .................. 122
4.6 Execution time of the ε-Constraints method on the different instances. ......................................................... 129

5.1 Overview of decentralised data centres. .......................... 137
5.2 Overview of E-GeNePi. ............................................. 142
5.3 Placement savings obtained by the Hill Climbing algorithm. . 150
5.4 Evolution of the average hypervolume of 10 runs during the execution of the different instances for the different algorithms. . 155

6.1 Overview of hybrid decentralised data centres. ................. 158
6.2 Overview of H2-D2. ............................................... 164
6.3 Computing the hypervolume and number of non-dominated solutions under the optimistic and pessimistic scenarios. . . . 169
6.4 Best, average and worst ranking for each algorithm on the ROADEF instances in terms of number of non-dominated solutions. . 174
6.5 Best, average and worst ranking for each algorithm on ROADEF instances in terms of obtained hypervolume. ................ 175
6.6 Evolution of the average hypervolume of 10 runs obtained using the different algorithms on the different instances over time when setting the VM prices to their average. ................ 179
LIST OF TABLES

2.1 Categorisation of the quality indicators with their requirements. 28
3.1 Characteristics of the different instances used in my evaluation. 82
3.2 Execution time of CPLEX for the resolution of the identity vector depending on the optimality tolerance gap. 85
4.1 Parameters for the different steps of GeNePi after a tuning study. 110
4.2 Summary of average solutions of 10 runs for the various algorithms and each of the used instances. 111
4.3 Summary of hypervolume of 10 runs for the various algorithms and each of the used instances. 112
4.4 Summary of the improvement obtained using GeNePi on both number of non-dominated solutions and hypervolume when applied on the different instances. 114
4.5 Average execution time of 10 runs of GeNePi and other evaluated algorithms on the different instances. 115
4.6 Summary of average results of 10 runs obtained with GeNePi, CPLEX, CPLEX combined with GeNePi, CBLNS, and CBLNS combined with GeNePi, against the initial assignment. 124
4.7 Average execution time of 10 runs of GeNePi, CPLEX, CPLEX combined with GeNePi, CBLNS, and CBLNS combined with GeNePi, against the initial assignment. 125
4.8 Average hypervolume and number of non-dominated solutions of 10 runs obtained with GeNePi, CPLEX+GeNePi, CBLNS+GeNePi, and e-Constraints method run for respectively 10, 20 and 30 days. 131
4.9 Average execution time of 10 runs of GeNePi, CPLEX+GeNePi, CBLNS+GeNePi, and e-Constraints method run for respectively 10, 20 and 30 days. 132
5.1 Instances size and allowed execution time. 146
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Summary of average solutions of 10 runs for the various algorithms and the various instances.</td>
<td>151</td>
</tr>
<tr>
<td>5.3</td>
<td>Summary of average hypervolume of 10 runs for the various algorithms and the various instances.</td>
<td>152</td>
</tr>
<tr>
<td>6.1</td>
<td>Characteristics of the different data centres and the allowed execution time for each of them.</td>
<td>167</td>
</tr>
<tr>
<td>6.2</td>
<td>Average results of 10 runs in terms of number of non-dominated solutions.</td>
<td>172</td>
</tr>
<tr>
<td>6.3</td>
<td>Average results of 10 runs in terms of hypervolume for both the optimistic and the pessimistic scenarios.</td>
<td>173</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td></td>
</tr>
<tr>
<td>BB</td>
<td>First Fit Descent Bin-Balancing</td>
<td></td>
</tr>
<tr>
<td>BSA</td>
<td>Biased Importance Sampling</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>Capital Allocator</td>
<td></td>
</tr>
<tr>
<td>CBLNS</td>
<td>Constraint-Based Large Neighborhood Search</td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>Cross-Entropy</td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>Constraint Programming</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
<td></td>
</tr>
<tr>
<td>DC</td>
<td>Data Centre</td>
<td></td>
</tr>
<tr>
<td>EURO</td>
<td>Association of European Operational Research Societies</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>First Fit</td>
<td></td>
</tr>
<tr>
<td>FFD</td>
<td>First Fit Decreasing</td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
<td></td>
</tr>
<tr>
<td>GRASP</td>
<td>Grasp greedy Randomized Adaptive Search</td>
<td></td>
</tr>
<tr>
<td>HC</td>
<td>Hill Climbing</td>
<td></td>
</tr>
<tr>
<td>HPC</td>
<td>High-Performance Computing</td>
<td></td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
<td></td>
</tr>
<tr>
<td>LNS</td>
<td>Large Neighborhood Search</td>
<td></td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed-Integer Linear Programming</td>
<td></td>
</tr>
<tr>
<td>MO</td>
<td>Multi-objective</td>
<td></td>
</tr>
<tr>
<td>MRP</td>
<td>Machine Reassignment Problem</td>
<td></td>
</tr>
<tr>
<td>NSGA</td>
<td>Non-dominated Sorting Genetic Algorithm</td>
<td></td>
</tr>
</tbody>
</table>
OS  Operating System
PaaS  Platform as a Service
PLS  Pareto Local Search
PM  Physical Machine
PSO  Particle Swarm Optimization
QoS  Quality of Service
RAM  Random-Access Memory
RF  Random Fit
ROADEF  French Operations Research & Decision Support Society
SA  Simulated Annealing
SaaS  Software as a Service
SLA  Service-Level Agreement
SPEA  Strength Pareto Evolutionary Algorithm
SQL  Structured Query Language
VC  Virtual Data Centre
VM  Virtual Machine
VNS  Variable Neighborhood Search
Submitted


Published


Data centres are facilities dedicated to providing, hosting and managing a large amount of 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: we store our data on-line, we stream our music, we use on-line banking, etc. Typically, data centres use virtualisation technologies extensively \[7\] to host their large number of services while maximising server utilisation. Virtualising resources means that data centres run most of their services on Virtual Machines (VMs) which can share the same\(^1\) Physical Machines (PMs), while ensuring the isolation of services (avoiding conflicts between workloads \[8\]). Given that VMs are ‘packed’ on PMs, optimising data centres is often seen as a problem where managers/algorithms have to find a better placement of VMs on PMs: move VMs on a smaller subset of PMs – decommissioning/stop PMs and automatically saving various costs (e.g., electricity and maintenance).

In this thesis, I model the reassignment problem for data centres: I define how VMs can be moved around to save some costs. I show that the problem is complex and requires many different actions and parameters to be taken into account.

\(^1\)E.g., VMware ESX accepts up to 320 VMs per host
1.1 Motivation

Data centres face a major problem which is the scale of the facilities (e.g., OVH, a European leader in the domain, have 260,000 servers in 17 distant locations, and host up to millions of services). At this scale, optimising a data centre becomes a challenge, especially given the different hard constraints and Service-Level Agreements (SLAs) which make the search space difficult to explore.

Data centres also evolve continuously: while new PMs are bought constantly (often more powerful and with different hardware settings than existing PMs), other ones get out dated or fail and are decommissioned. Besides, applications are deployed and released all the time, which leads to server sprawl; a situation where the under-utilised servers take up more resources (space, energy and human power) than what is required to run the workload. Moreover, managers may want to increase the reliability of their data centres and move the workload from overloaded PMs to less loaded and/or more powerful ones. Finally, they may also try to move the workload to power efficient PMs, in order to lower the cost and the environmental impact of their data centres.

Finally, large companies’ data centres are made of a number of distributed and relatively autonomous smaller data centres. Large modern organisations often face internal segmentation, based on geography, legal jurisdictions, age/maturity of lines of business, and sheer size. This process, sometimes referred to as siloing, means that the organisation has employees who identify themselves with their group rather than with the whole organisation, and who view their group’s objectives as being more relevant than the organisation’s objectives. Hence, capital allocators (CAs) of the different hosting departments sometimes compete or at least have different perspectives on the best way of making the system better. It is crucial for large organisations to address the difference in objectives and incentives between capital allocators, and in many large organisations, it is seen as one of the biggest barriers to optimisation.

---

2. A Service-Level Agreement is an explicit commitment between a service provider and a customer. In the context of data centres, SLAs often describe elements such as the amount of resources allocated, the performance guaranteed, etc.
1.1 Motivation

Additionally, there has been a proliferation of cloud services in the past years, from VMs of different flavours to out-of-the-box platforms (e.g., ready to use Machine Learning tools). The many benefits of these cloud solutions [13], including but not limited to their cost, have accelerated the adoption of the Cloud for all sorts of companies. However, modern large and often global organisations tend to ponder over outsourcing to the Cloud more than small and medium companies [14], with only 17% of them reported having 1000+ VMs in the Cloud. Some of the many reasons for this slow process are the complexity of their software systems [15], the types of their workloads and the privacy/security of their data and products – as well as the distribution and segmentation of the data centres (DCs) of these large and global companies [5]. However, the hybrid cloud solution [16], i.e., mixing private infrastructure and public cloud services, is now seen as a potential option for these large companies [14] as it gives them the benefits of both worlds. On one hand, the Cloud provides quick infrastructure provisioning and deployment [17], while on the other hand, companies can still maintain their own infrastructure when exact characteristics of servers, performances and reliability are important [18].

All these elements: scale, evolution, autonomy of hosting departments and public clouds, define a complex search space that has been at the centre of a lot of attention in the recent (and not so recent) past (see Chapter 2) as potential cost savings are huge. A survey conducted by Anthesis Group and Stanford University in June 2015 revealed that about 30% of servers are sitting “comatose” (about 10 million comatose servers worldwide) without receiving any query or delivering any information for more than six months, which translates to a capital of $30 billion sitting idle globally.

On top of all that, the notion of best reassignment is controversial, as it is more about the right balance between the different objectives that will be favoured and improving the performance of one objective usually leads to a worsening of other ones. Decision makers in large organisations tend to favour tools that allow them to manipulate good reassignments, i.e., possible solutions that are better than every other possible reassignment on a particular combination of objectives. Eventually, decision makers can look at the different possible reassignments and make a decision based on local optimisation (e.g., favouring objective 1 which gives a bigger gain than objective 2 while the latter is usually more important).
1.2 Problem Definition

My thesis addresses the problem of VM Reassignment in Large Data Centres, i.e., where scale, multiplicity of optimisation objectives, distribution and autonomy of hosting departments and introduction of cloud offers are crucial. In this section, I describe each of the elements of the problem as defined in the previous sentence.

The VM Reassignment Problem consists in assigning each VM in the system to a PM (either a new one or the one it is already on) according to some objective function that I try to optimise (typically minimise). Any reassignment has to satisfy (a large number of) constraints defined by the system. In some cases, VMs do not move during the reassignment, being assigned to the PMs in which they are already hosted. Every PM has several resources (e.g., CPU, RAM and disk) in limited capacities. The reassignment of a VM is achieved using a live migration, meaning that the VM is transferred while it is still running on the initial PM. I also consider that the quantity of resources that the VM needs is fixed and corresponds to the VM parameters/SLAs. Note that some resources are called transient (e.g., RAM and disk), i.e., they are needed on the initial and final PMs during a migration, as the VMs use the resources on both PMs during the reassignment.

The Multi-objective VM Reassignment Problem consists in optimising the data centres while considering more than one objective, i.e., several independent objective functions. The goal here is to search for optimal (or nearly optimal) solutions that consider a trade-off between two or more conflicting objectives. In other words, the Multi-objective VM Reassignment Problem tries to find non-dominated solutions (better than every other solution in the search space on at least one of the objectives). I will describe what this means in detail in the following chapters (in particular in Chapter 2).

In a decentralised data centre, the infrastructure is composed of a set of hosting departments with a certain degree of autonomy in the VM placements they favour, with CAs competing or having different perspectives/preferences on what makes a VM placement better than another. The Multi-objective VM Reassignment Problem in such infrastructure is composed of two different problems: (i) offering top-level decision makers a panel of possible and
non-dominated VM reassignments to the hosting departments, and (ii) placing the VMs assigned to a hosting department according to individual CAs’ preferences.

The Multi-objective VM Reassignment Problem in a hybrid cloud, mixing private decentralised infrastructure and public cloud, extends the former by adding cloud VMs of different flavours and price. Given the variability in the public cloud pricing and unlike the objectives considered in the previous context, the cost of hosting the VMs in the Cloud is expressed as an interval and therefore I use dominance for solutions with interval objectives.

1.3 Thesis Approach

During this PhD, I have taken following approach and I have addressed my problem in three phases of increasing complexities.

First, this is a very practical problem, so I have taken a practitioner’s perspective on the background and challenges. I have worked with collaborators in IBM (frequent discussions and five co-authored papers) and Bell Labs (three-month internship and one paper in the making). I have identified from these collaborations what the relevant practical challenges are and how they would be addressed in the industry. In particular, the multi-objective, the hybrid and the decentralised natures of this problem have been identified as key elements of any practical solution. The scale is another element that my collaborations with industry have targeted as important.

Given the complexity of the data centres I am dealing with in this thesis (large scale, several hard constraints, decentralised and hybridised), I take an incremental approach, presenting problems and contexts of increasing complexities (see Figure 1.1). I start with a centralised data centre: the most basic structure that has been studied extensively in the literature, usually with a single objective formulation (see Chapter 2). In this context, the data centre is seen as a unique structure with decision makers having a full control over its (potentially geographically diverse) elements. I then focus on decentralised data centres where local capital allocators have different preferences for their own hosting departments and only partially share the view of their managers. Finally, I introduce cloud offers in the decision making process, creating a
1.3 Thesis Approach

Figure 1.1: Thesis approach: from a ‘single’ centralised data centre\(^{(1)}\), a decentralised data centre\(^{(2)}\), to a hybrid data centre\(^{(3)}\).

large panel of possible reassignments for each VM, internally (in one of the hosting departments) and externally in the Cloud.

To address the problem of multi-objective VM reassignment in large centralised data centres, I first study the particularities of the classical optimisation techniques that have been proposed to the mono-objective problem (the most studied problem) and I evaluate what makes them suitable. Next, I study the applicability of exact solutions such as Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) techniques, and I evaluate the performance of their extension to the multi-objective case (i.e., my case). I also perform a thorough evaluation of state-of-the-art metaheuristics with different optimisation strategies (e.g., genetic algorithm, local search, greedy) in the field of multi-objective optimisation. In addition, I study the possible contribution of hybridisation of both metaheuristic + metaheuristic, and exact + metaheuristic on the problem. Afterwards, I include an additional challenge in the problem and consider the decentralised aspect of data centres. To address the decentralised context, I propose a two-level system that (i) satisfies individual CAs’ placement preferences, (ii) offers top-level decision makers with a panel of possible good reassignments that they can navigate, and (iii)
scales up to large data centres. Lastly, I expand the problem of decentralised data centres to consider the reassignment of VMs to available public cloud locations as a possible option and extend the two-level system to encompass public clouds as a possibility when reassigning VMs, providing payment of a certain cost.

In my experimental study, I use a dataset that is widely used in the Operations Research community, for the evaluation of VM reassignment solutions, that was proposed by Google to a joint challenge from ROADEF\(^1\) and EURO\(^2\) societies \([19]\) in 2012. 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 I adapt the instances to fit my scenario. My instances were generated in tight collaboration with one of my collaborators, J. Thorburn, who works on assets utilisation efficiency in IBM data centres.

1.4 Contributions

The main contributions of this thesis are the following:

- I formally define the multi-objective VM reassignment problem for large data centres, decentralised data centres and hybrid decentralised data centres.
- I design, implement and evaluate the performance of my three-step hybrid metaheuristic (i.e., GeNePi) for the multi-objective VM reassignment in large data centres, which outperforms the state-of-the-art techniques quantitatively and qualitatively.
- I design and implement a two-level system for the multi-objective VM reassignment in the context of both large decentralised data centres and large hybrid decentralised data centres.

---

\(^{1}\)ROADEF: the French Operations Research society

\(^{2}\)EURO: the Association of European Operational Research Societies within IFORS; the ‘International Federation of Operational Research Societies’
This thesis also includes the following supporting contributions:

- I model the multi-objective VM reassignment in centralised data centres as a linear problem and evaluate the performance of linear solvers and hybrid techniques on the problem.

- I adapt my solution (GeNePi) to optimise the multi-objective VM reassignment in both large decentralised data centres and large hybrid decentralised data centres.

- I extend GeNePi to handle objectives with interval values.

1.5 Thesis Structure

The rest of this thesis is structured as follows:

- Chapter 2 describes some background information useful for the understanding of this thesis and presents the related work in the fields of single and multi-objective VM reassignment in both centralised and distributed data centres.

- Chapter 3 presents the multi-objective VM reassignment problem in centralised data centres and studies the applicability of linear solvers and best algorithms in the mono-objective case. I start by providing a full linear formulation for the multi-objective VM reassignment problem in centralised data centres. Then, I study the best way for using both an MILP solver (i.e., CPLEX) and a state-of-the-art Constraint-Based hybrid algorithm (i.e., CBLNS).

- Chapter 4 describes my solution to the multi-objective VM reassignment problem in centralised data centres. I start by detailing my three-step method; GeNePi. Next, I perform a thorough evaluation against the state-of-the-art. Then, I propose a hybrid technique that combines either CPLEX or CBLNS with GeNePi to improve the performance while keeping the execution time low. Finally, I assess the quality of my algorithms in comparison to an exact algorithm run for up to a month.
1.5 Thesis Structure

- Chapter 5 presents the multi-objective VM reassignment in decentralised data centres. I start by providing the first full formulation of the multi-objective VM reassignment for decentralised data centres as a two-level problem. Next, I detail the two-level system that I designed to optimise the problem with the algorithms run at each level. Last, I perform a thorough evaluation of the different components of my system.

- Chapter 6 presents the multi-objective VM reassignment in hybrid and decentralised data centres. I start by providing a formal definition of the problem. Then, I detail the two-level system that optimises the VM reassignment in such an environment with the different algorithms run at each level.

- Finally, in Chapter 7, I present my conclusions and ideas which could lead to interesting future contributions in multi-objective VM reassignment in large data centres.
2.1 Background

In this section, I describe some background of my research in three parts: (i) virtualisation (ii) data centre consolidation through VM placement and (iii) multi-objective optimisation.

2.1.1 Virtualisation

Virtualisation is one of the enabling technologies behind cloud computing. It allows a partitioning of the physical machine resource capabilities, provides computation platforms that are both flexible and scalable [7] and permits an efficient use of physical resources by consolidating several applications into the same large physical machine while guarantying their isolation from each other. Virtualisation also abstracts details of the hardware and provides the applications sitting on top of them with virtualised resources. Having the virtualisation technology responsible for the mapping between the virtual and physical resources takes out the worry of what hardware an application would be running on and makes it possible to seamlessly move applications from
2.1 Background

one PM to another. There exist two major virtualisation techniques: Virtual Machines and Software Containers.

2.1.1 Virtual Machines

Virtual Machines (VMs) are self-contained, isolated and independent software containers and can be considered as separate computation environments (similar to an environment provided by a physical machine, from the point of view of the application running on top of it). In addition to allowing several applications to run on the same physical host in total isolation from each other, every VM runs its own operating system (OS), which can be different from the other VMs and also different from the one on top of the physical machine.

There exists several techniques that enable the creation and management of virtual machines [20] and at least one of them is required for that purpose. The most popular and well-known amongst them are hypervisor-based (e.g., VMware, Xen and KVM), which use a thin layer software (i.e., the virtual machine monitor or the hypervisor) to dynamically allocate resources to VMs and map their instructions to the host’s hardware.

2.1.2 Software Containers

The other approach for virtualisation is based on containers. In this virtualisation technology, the host’s operating system allows the generation of several isolated user spaces (each with its own environment variables and data) instead of a unique one as it is the case in standard OSs (modern OSs usually partition the memory into two partitions: one for the kernel and another one for the users). As is the case with VMs, containers also seem like real physical machines from the perspective of the applications sitting on top of them. Each of the containers run as a process on the host’s OS and does not require an additional OS, making it both lightweight and fast to start.

There are several container-based virtualisation systems, most of them being free projects (e.g., Linux Containers, OpenVZ and Linux-VServer) and have near-native CPU, RAM and disk performances. However, they differ on
their resource management as some of them introduce new resource limiting policies that go beyond the Linux standard (e.g., limiting the number of processes able to run on a single container). In parallel to the virtualisation system, there is usually an orchestrator responsible for quickly building and distributing applications. The most famous amongst them is Docker which usually uses LinuX Containers as the underlying virtualisation system.

2.1.1.3 Virtual Machines vs. Software Containers

The difference between hypervisor- and container-based technologies can be seen in Figure 2.1. We see that every VM uses its own OS and runs its application on top of it, whereas in containers, applications get an immediate abstraction of the host OS as they are linked to it directly through the virtualisation layer. Thanks to this immediate abstraction [21], containers reduce process and storage overheads that are common with VMs [22], and given their lightweight, they can achieve higher densities than VMs and decrease the start-up time [23]. However, given that containers do not have their own OSs and use the one on the host, this reduces the interoperability (the application on the container only works with that type of OSs, which have to be a Linux based one). Furthermore, having containers sharing the kernel of the host OS raises some security issues and makes containers less isolated and thus riskier than VMs. In addition, the lack of standardisation in container-based virtualisation and the lack of standard monitoring services prevent companies from using them [23]. Based on these aspects, I consider in this thesis that virtualisation is performed by means of hypervisors with applications sitting on top of VMs.

2.1.2 Consolidation Through VM Placement

Data centre consolidation is an important objective for any DC manager. There exists several ways for consolidating data centres. Managers of DCs can upgrade their systems (e.g., servers, cooling and networking) to more recent/efficient technologies, or move their DCs into more convenient real estates. However, the focus in this case is on better placing VMs. VM placement
2.1 Background

Figure 2.1: Comparison of hypervisor- and container-based virtualisation technologies.

is an extension of the $d$ – *dimensional* bin packing problem. While the $d$-dimensional bin packing problem aims at reducing the number of bins, the VM placement problem considers moving VMs between an already given set of bins (a.k.a., PMs). It tries to optimise different goals while integrating many topic-related constraints. I study here two of its variations: VM assignment and VM reassignment.

### 2.1.2.1 d-Dimensional Bin Packing

The $d$-dimensional bin packing consists of packing a set of items of various sizes, in the least number of homogeneous bins. This problem has two main variants: geometric and vector packing. In geometric bin packing, items can be rotated in the space before being placed in the bins. In the case of processes and servers though, each dimension of the space has a meaning and it is not possible. Vector packing has been a very popular challenge in computer science and engineering, for instance, in the system domain [24].

In the context of 2-dimensional bin packing, Caprara and Toth [25] developed a set of heuristics and exact solutions, which were since outperformed by an approximation algorithm proposed by Kellerer and Kotov [26] with a performance ratio of 2 in the worst-case. Concerning problems with more than
2.1 Background

2 resources, Beck and Siewiorek [27] modelled and made a thorough evaluation of different algorithms for assigning tasks in a multiprocessor computer. Whereas Leinberger et al. [28] addressed the same problem, but for systems running tasks in parallel. Jansen and Öhring [29] studied d-dimensional bin packing with conflicting items and proposed some approximation algorithms, while Gendreau et al. [30] put forwards several heuristics and lower bounds which take into consideration this set of constraints. Most recent works deal with the vector d-dimensional bin packing for storing multi-media content. They either study different algorithms (e.g., Panigrahy et al. [31] evaluate the performance of various First Fit algorithms) or try to find better approximation algorithms with mathematically proved lower bounds (e.g., Shachnai and Tamir [32]).

2.1.2.2 VM Assignment

The assignment of VMs consists of finding an initial placement for one or several VMs in a set of physical machines. It is also referred to as VM allocation, VM placement or sometimes as resource provisioning. The assignment is done with respect to different sets of constraints that may vary from a facility to another. These constraints depend on the topology (e.g., limits in the network connections, PMs organised by racks, etc.) and on the SLAs signed with customers. The specificity of this placement is that VMs are considered as initially off. The only concern here is the final placement of VMs and not the migration.

Early works done in this area are more theoretical. They formulate the problem as a bin packing problem and propose simple heuristics and rules. Gupta et al. [33] propose to combine two heuristics; each addressing a specific set of constraints (i.e., VM to VM and VM to Server incompatibilities). Some works study different First Fit algorithms. Dósa [34] describe the performance of the First Fit Decreasing algorithm, and Bansal et al. [35] compare different First Fit algorithms. Other works combine heuristics with greedy techniques such as First Fit and Best Fit Decreasing algorithms. Panigrahy et al. [31] study different First Fit algorithms and put forwards a new algorithm based on a geometric heuristic able to scale to data centres of large sizes while maintaining its performance level. Wilcox et al. [36] changed this trend by applying a Reordering Grouping Genetic Algorithm (RGGA).
2.1 Background

2.1.2.3 VM Reassignment

Unlike in the VM assignment, it is considered here that VMs are already running on PMs and the aim is to reallocate them without stopping the service by means of live migration (see below). Two cases can be considered in this context: the first case is a reactive VM reassignment, and the second case (i.e., the one addressed in this thesis) is a planning of the VM reassignment.

**Live Migration** of VMs from their initial hosts to other ones allows the management and the consolidation of resource allocations without the need to stop the services running on top of these VMs [37]. Live migration is key when it comes to VM reassignment [38]. For instance, when the load is low, some VMs can be placed on the same PM so that other PMs can be turned off, consequently reducing servers sprawl and resource under-utilisation. With the increase in workload, VMs can be spread over PMs with a lower load in order to avoid the violation of some SLAs.

However, live migration brings also some negative aspects as it takes time and adds an extra load to the network and to the PMs involved, which can impact negatively some SLAs [39]. It is difficult to understand the exact effect of live migration, but many works aim at modelling them. Verma et al. [40] found that live migration only increases the load on the initial host and presented a plausible model for the prediction of resource overheads led by live migration [41]. Many works are looking for a more detailed and more universal model [42–44]. In addition to the resource consumption, live migration also necessitates a non-negligible amount of energy and a significant bandwidth, which led to works considering the optimisation of live migration side effects as objectives in their models [45, 46].

Different live migration strategies exist in the literature and are surveyed in the work of Pasumarthy [47]. Currently, the technique that is the most adopted by virtualisation systems is called the pre-copy strategy. This technique starts by transferring a snapshot of the VM while this one is still running, before entering an iterative push phase where the updates (i.e., modifications during the transfer) are transmitted again.

Live migration used to be done only within boundaries of the same DC. However, with the development of multiple cloud architectures (such as those
2.1 Background

described by Grozev and Buyya [48]) and tools to abstract the differences that might exist between them (e.g., jClouds and Libcloud), in addition to the huge advances in standardisation and the efforts to make live migration secure [49], we see an increase in interoperability and thus live migration is becoming even performed between different DCs.

Reactive VM Reassignment where the system constantly optimises placement of VMs by migrating them in reaction to some changes and as long as it brings some improvement. Bobroff et al. [50] try to reduce the number of running PMs and develop a First Fit algorithm which dynamically reassigns VMs to PMs that are able to host them without violating their SLAs. Jung et al. [51] design a hybrid approach that uses modelling and optimisation offline to generate suitable configurations, which are then encoded as adaptation policies that are used at run-time. They also demonstrate their algorithm in a real environment that requires an on-line consolidation of servers hosting multiple multi-tier applications. Li et al. [52] propose EnaCloud; an approach which allows a dynamic and live reassignment of applications sitting on top of VMs with a consideration of the energy efficiency through a reduction in the number of running PMs. The reactive VM reassignment has proved some performance, but it is very complicated to implement on data centres, because of the hardness of collecting data from the whole infrastructure (usually from different locations) and quickly analysing them using several metrics. The use of such techniques is also very risky to implement on a real data centre, as every modification generates a risk of failure and decreases the reliability.

Planning VM Reassignment is allowed a longer execution time to optimise the current VM placement and provide the decision maker with suitable VM reassignment candidates. This allows a large probing of the possible reassignments to get as much improvement as possible. Once the new placement is chosen, it is going to be applied for a long period (e.g., a week or a month), without being changed; or at most with minor changes. Therefore, the quality of the found solution and its resiliency to the changes that may occur before the next reassignment are important to include as objective functions.
2.1 Background

2.1.3 Public Clouds

Public clouds are data centres that make different services (e.g., applications and storage) available to the general public in a virtualised form and accessible over a public network (e.g., the Internet). The most noticeable difference between public and Enterprise data centres (a.k.a., private data centres) is that in the latter, only individuals from the organisation are allowed to access and run services on the infrastructure, whereas in the former, multiple external clients/companies are allowed to leverage it. These customers often have some requirements regarding the quality of service. They may also subscribe to extra features such as Elasticity (i.e., adaptation of the allocated resources to meet peaks in demand). According to a report from RightScale in 2017 [53], the public cloud market is largely dominated by Amazon (AWS), followed by Microsoft Azure and Google Cloud. Public clouds use different service models. There are three most commonly used ones in addition to other derivatives and domain specific ones: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

2.1.3.1 Software as a Service

Companies can allow their customers to use their software, by deploying them on the Cloud (a hosting environment) and making them available as a service through a web interface (e.g., web browser, an application or a personal digital assistant). In addition to running their services, these companies are also often in charge of their optimisation through scaling. They ensure the availability of the services, their security and recovery in case of a disaster. However, SaaS do not offer customers control over the infrastructure of the hosting environment. Examples of such SaaS include Gmail, Google Drive, SalesForce.com, etc. This type of services is popular among people whom would like to use an application without bothering themselves with the details behind them.

2.1.3.2 Platform as a Service

PaaS are software development platforms with a full ‘software lifecycle’ support, deployed on the Cloud, and made available to customers through the
2.1 Background

Internet, so that they could develop their own applications (e.g., a SaaS) on them. The difference between PaaS and SaaS is that the latter only hosts and gives access to a working service, whereas the former allows the development and the deployment of applications as SaaS. This service is mostly popular among developers whom would like to create their applications, as PaaS allows them to focus more on the development aspects and takes away the worry about the management and execution of their applications. Examples of PaaS include Amazon AWS, IBM Bluemix (also supports private PaaS) and Microsoft Azure ML (specialised in Machine Learning).

2.1.3.3 Infrastructure as a Service

As its name suggests it, IaaS provides the customer of the Cloud with the computing infrastructure, mostly virtualised (i.e., as virtual machines). Additionally, depending on the subscribed plan, it may also offer storage, virtual Local Area Network, IP address, firewalls, etc. The customers can run their own applications on the virtual machines without worrying about the underlying physical host, and in isolation of other VMs they might share the host with. Examples of IaaS include Amazon EC2, Windows Azure, Google Compute Engine, Rackspace, etc.

2.1.3.4 Domain Specific Services

There exist different new service names that are more domain oriented, such as Database as a Service where virtualised storage of different types (SQL and NoSQL or centralised and distributed) is delivered to clients (e.g., Amazon S3, Google Spanner and Google Bigtable). However, they are all descendant of the three aforementioned main services. For instance, Database as a Service and Network Functions as a Service are SaaS, Machine Learning as a Service is a PaaS, and Containers as a Service is an IaaS.

2.1.4 Resolution vs. Optimisation

Multi-objective optimisation is the art of optimising a problem having two or more objective functions. The aim is to provide a set of solutions optimising
2.1 Background

the different objectives. This set is called \textit{non-dominated set} or \textit{Pareto frontier} and all the solutions that belong to this set should be better than any others in at least one objective. Depending on the size of the problem, its complexity and the goal behind the optimisation, two techniques may be applied: (i) a multi-objective resolution or (ii) a multi-objective optimisation.

2.1.4.1 Multi-objective Resolution

The aim of the multi-objective resolution is to get all optimal non-dominated solutions (Pareto optimal), which are solutions that can not be strictly dominated by others. 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. [54] 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. Several methods have been developed to solve efficiently multi-objective problems and they are organised in the work of Ehrgott et al. [55] into four categories:

\textbf{Extending Mono-objective Algorithms} when the problems can efficiently be solved with one single objective in a polynomial time and when the extension to multiple objectives is possible. These methods include labelling algorithms that find the shortest path such as Dijkstra’s and Belman’s algorithms for which extensions to the multi-objective problem exist and are compared by Raith and Ehrgott [56]. Extending mono-objective algorithms also include greedy algorithms that generalise algorithms of specific problems such as Prim’s and Kruskal’s for the spanning tree. Greedy algorithms have also been developed by Gorski et al. [57] to a class of knapsack problems with three objectives.

\textbf{Scalarisation} where the multi-objective problem is transformed into several mono-objective ones that are solved to optimality. There exist different scalarisation techniques [55], which in addition to finding optimal non-dominated
solutions, achieve different properties [58]. Most popular scalarisation techniques include:

- **Weighted Sum** [59, 60] where the objectives are successively aggregated using a weighted sum with weight vectors the are specifically designed to methodically cover the whole convex hull. This scalarisation has the advantage of not adding any constraint to the original problem (thus maintaining any existing pattern and possible linearity of the mono-objective problem that could help in the resolution). However, it only allows finding solutions on the convex hull and not the non-dominated solutions within the concave hull.

- **\( \epsilon \)-Constraints** [61, 62] method transforms the 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 \( \mathcal{E} \) (a different one for every mono-objective problem). Despite adding extra constraints to the original problem, \( \epsilon \)-Constraints method maintains its possible linear formulation and most importantly allows finding the exhaustive (maximal) optimal Pareto front.

- **Weighted Chebychev** [63] which aggregates the objectives using different fine tunings of the weighted Chebychev norm to find all non-dominated solutions, which maintains the structure of the original problem, but due to the complexity of the Chebychev norm, the computation takes a hit and the optimisation of the mono-objective problems become more complicated.

**Two-Phase Method** is motivated by the fact that in absence of adaptation of mono-objective algorithms to solve the multi-objective problem (e.g., labelling or greedy algorithms), it is desirable to repeatedly solve the mono-objective problems with the same computation complexity as the original problem to get as many solutions as possible. The weighted sum seems ideal for that as it maintains the computability of the original problem. However, given that the weighted sum allows only to get the solutions in the convex hull, it has to be augmented using some enumeration technique to get solutions not belonging to it (the second phase). The two-phase method is exposed in details by Przybylski et al. [64].
2.1 Background

**Branch and Bound** is one of the most common methods when solving combinatorial and Mixed-Integer Linear Programs. As it is indicated in its name, it is based on: (i) Branching: where a search tree is split into different branches with a decision variable taking a subset of its domain of values, and (ii) Bounding: where the method tries to get a good lower bound for the given branch and decides whether it is worth investigating or not. Several extensions of the Branch and Bound method have been proposed for multi-objective problems [55, 65, 66] and they differ on their computation of the lower bounds, branching heuristics and exploration strategies.

2.1.4.2 Multi-objective Optimisation

These are the most frequently employed techniques when dealing with complex and large problems such as large scale data centres. Several techniques have been created over the last years to optimise complex problems and find non-dominated solutions [67]. The literature in the matter is very diverse and hard to compare as algorithms usually address different problems from each other. The best performing techniques are from the metaheuristics family [68, 69] and range from Genetic Algorithms, to Particle Swarm Optimization, to Variable Neighborhood Search, etc. Many definitions of metaheuristics can be found in the literature and most of them agree that a metaheuristic should have a number of basic properties [70]:

- Non-exact techniques.
- Aim at exploring efficiently the search space to find near-optimal solutions.
- General strategies and templates that guide the exploration of the search space.
- May use some knowledge specific to a problem implemented as heuristics controlled by the general strategy.

There are several criteria that allow the clustering of metaheuristics. One of the most intuitive and widely accepted categorisations between metaheuristics distinguishes between the origin of these algorithms as either nature or non-nature inspired [68, 70]:

2.1 Background

Nature Inspired such as:

- Simulated Annealing (SA): a method that mimics the process of annealing in metallurgy: heating a material to melt and shape it (often iron in a foundry) and slowly cooling it down to decrease its defects and make it more robust. SA starts with an initial solution and iteratively moves from its current solution to another one in its neighbourhood, providing that it does not worsen its fitness function more than what is allowed by a probability function. This probability function has often a similar shape than the cooling in the annealing, which leads to a lesser acceptance of solutions that worsen the fitness function over time.

- Swarm Intelligence: are techniques that are motivated by the fact that single animals (e.g., ant, bee, bird, fish, etc.) are not smart on their own, but their colonies/flocks/schools are. Therefore, studying swarm intelligence can help manage and optimise complex systems. Most well-known optimisation algorithms based on swarm intelligence include the Particle Swarm Optimization (PSO) and the Ant Colony Optimization (ACO).

- Evolutionary Algorithms: with the most popular being Genetic Algorithms (GA) are inspired by the evolution in biology and use mechanisms such as reproduction (a.k.a. crossover) and mutation to evolve a set of solutions called a population or a generation.

- Estimation of Distribution (ED) is a metaheuristic that guides the optimisation by selecting promising candidate solutions and sampling them based on explicit and iteratively updated probabilistic models.

Non-Nature Inspired Techniques including:

- Greedy Randomized Adaptive Search Procedure (GRASP) consists of a constructive and a local search phase. The constructive phase allows the creation of an initial solution that is both of a good quality (the greedy aspect) and that avoids local minima (through randomisation). Then, the initial solution is improved using a local search.
2.1 Background

- Variable Neighbourhood Search (VNS) is a local search that combines two types of neighbourhoods: the first neighbourhood is for improving the current solution until reaching a local optimum and the second one aims at introducing a perturbation to get out of the local optimum.

- Tabu Search (TS) is also a local search that moves from the current solution to its best neighbour. However, TS allows the move even if the neighbour worsens the fitness function. To avoid cycling between the same solutions, a short-term Tabu list (a list of forbidden moves) is updated to restrict from going back to a previously visited solution.

- Path Relinking (PR) explores for any given two solutions the intermediary solutions between them with the aim of finding solutions that combine their both qualities. This goes by making small moves from one of the solutions to eventually changing it into the second one.

Amongst the large number of existing metaheuristics, evolutionary algorithms have proved their performance against the others on multi-objective problems and are popular among researchers and practitioners [71]. Various genetic algorithms with different operators have been developed and studied over time. The first evolutionary algorithms were proposed more than two decades ago. They are simplistic genetic algorithms which are often categorised as non-elitist such as MOGA [72] and NPGA [73]. Then, around the year 2000, evolutionary algorithms with a higher performance (labelled elitist) were proposed such as NSGA-II by Deb et al. [74] and SPEA2 by Zitzler et al. [75]. These algorithms use an elite preservation strategy, combined with a fitness function based on diversity maintenance and Pareto dominance, which allow them to achieve a clear improvement in performance [76].

In the recent years, some works extended their problems to a large number of objectives (sometimes more than 10). These problems are called many-objective problems. Some recent studies from Purshouse and Fleming [77] and Ishibuchi et al. [78] show that a large number of objectives decreases the dominance likelihood between solutions, thus drastically impacting the performance of dominance-based evolutionary algorithms. In response to these studies, very recent algorithms try to factor that into account such as NSGA-III by Jain and Deb [79], HypE by Bader and Zitzler [80] and SMS-EMOA by Beume et al. [81].
2.1 Background

2.1.5 Evaluation of Multi-objective Optimisation Techniques

Creating algorithms that approximate the Pareto frontier is an easy task. However, proving that they are better than others is an issue as there is no standard and general metric to evaluate them. Unlike the resolution where methods find ‘same’ results and where the comparison is done regarding the spatial (i.e., occupied space in memory) and temporal (i.e., execution time) criteria, evaluating the performance of two non-exact methods is harder as all of them may find different sets of solutions. Some algorithms aim at finding a set of solutions that make a trade-off between the objectives, while others aim at finding solutions which optimise them separately.

Comparing multi-objective optimisation approaches is complex as the set of solutions they give on a problem can be seen from different perspectives: coverage, closeness to the Pareto frontier, variety, and many more [82]. 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.

Several metrics and numerous operators have been designed in order to allow a comparison between multi-objective optimisation algorithms. However, each of them handles only some perspectives (e.g., number of solutions, spread on the Pareto frontier, etc.). Thus, depending on the context and the expected solutions (wanted by the decision maker), different metrics may be used. Existing metrics aim at evaluating one of two goals: (i) quality of solutions: how good is the obtained Pareto front? or (ii) diversity of solutions: how large/representative are its solutions?

2.1.5.1 Quality Metrics

Quality metrics are used to assess the quality of Pareto front solutions obtained by the different algorithms.

**Hypervolume (HV)** is the most used metric amongst the multi-objective optimisation community for comparing different sets of non-dominated solutions [83]. The intuition behind the hypervolume [76] is that it gives the volume (defined in the k dimensions of the search space) dominated by the Pareto
2.1 Background

The hypervolume is the area between the solutions and the reference point. The reference point represents the worst possible value for each objective. The obtained measure represents the area covered by the approximate Pareto front: the higher the better. The bigger the hypervolume, the more interesting are the solutions in the found non-dominated solution set, as they increase the dominated area. In formal terms, this is proven by Fleischer \cite{84} who states that the maximisation of the hypervolume is equivalent to finding the optimal Pareto frontier. In a more formal way, the hypervolume is defined as follows:

Let $A$ be the set of points on the Pareto front and $O_i(s)$ be the value of the objective $i$ of the solution $s \in A$, then the hypervolume of $A$ is represented by:

$$HV(A) = \lambda \left( \bigcup_{s \in A} [O_1(s), r_1] \times \ldots \times [O_k(s), r_k] \right)$$  \hspace{1cm} (2.1)

where: $\lambda$ is the Lebesgue measure \cite{85}, $k$ is the number of objectives, $[r_1, ..., r_k]$ is a reference point taken far from it and $[O_1(s), r_1] \times \ldots \times [O_k(s), r_k]$ is the k-dimensional hyper cuboid consisting of all points that are weakly dominated by the point $s$ but not weakly dominated by the reference point.

**Inverted Generation Distance (IGD)** measures the average of how far is every solution $s$ in the reference front $R$ (a set of solutions that are never dominated by the obtained non-dominated solutions) to its closest solution in the set of non-dominated solutions $A$ \cite{86} $d(s, A)$. IGD is complementary of the $\epsilon$ metric in its way of evaluating the distance between the Pareto and the reference fronts. The lower the IGD the better the Pareto front.

$$IGD(A, R) = \frac{\sum_{s \in R} d(s, A)}{|A|}$$  \hspace{1cm} (2.2)

**Generation Distance (GD)** is the ‘reverse’ of the IGD as it measures the average of how far is every solution $s$ in the set of non-dominated solutions $A$ from its closest solution in the reference Pareto front $R$ \cite{87} $d(s, R)$. The lower the GD the better is the set of non-dominated solutions.

$$GD(A, R) = \frac{\sum_{s \in A} d(s, R)}{|R|}$$  \hspace{1cm} (2.3)
2.1 Background

Euclidean Distance from the Ideal Solution (ED) measures the Euclidean distance \( d \) between the Ideal point (a point with the best value for every single objective – which most of the time does not correspond to any feasible solution) and the closest solution to it from the obtained non-dominated set. The smaller this distance the better. The Euclidean Distance between a non-dominated set \( A \) and an Ideal point \( I \) is defined as:

\[
ED(A, I) = \min_{s \in A} (d(s, I)) \tag{2.4}
\]

Epsilon (\( \epsilon \)) measures the shortest distance that is required to transform every solution in a Pareto front \( A \) to dominate the reference front \( R \) [88]. The lower the \( \epsilon \) value the better is the Pareto front. The \( \epsilon \) metric finds the smallest multiplier \( \epsilon \) such that every solution \( R \) is dominated by at least one solution in \( A \), and is defined as:

\[
\epsilon(A, R) = \min(\epsilon) \mid \forall s \in R, \exists s' \in A, s \geq \epsilon s' \tag{2.5}
\]

2.1.5.2 Diversity Metrics

Diversity metrics ensure that the set of solutions that are presented to the decision maker in order to choose from is diverse.

Number of Non-dominated Solutions (\#Sol) corresponds to the quantity of solutions that are not dominated in a set of solutions \( A \). This is obtained by counting the number of non-dominated solutions in the population of every generation. The higher this number the better as it means that more choices are provided to the decision makers to navigate and to select amongst them.

\[
\#Sol(A) = |R| \tag{2.6}
\]
2.1 Background

**Generated Spread (GS)** measures the solutions extent spread in the Pareto front and evaluates their distribution [89]. The lower the generated spread the more diverse if the Pareto front (i.e., the better). For a set of solutions $A$ and a reference front $R$:

$$ GS(A) = \frac{\sum_{k=1}^{m} d(e_k, A) + \sum_{s \in A} |d(s, A) - d_{avg}|}{\sum_{k=1}^{d(e_k, A)} + |A| \times d_{avg}} $$

(2.7)

with $e_k, k \in \{1, .., m\}$ are the extreme points in $R$, $d(s, A)$ the distance between the point $s$ and the set $A$, and $d_{avg} = \frac{\sum_{s \in A} d(s, A)}{|A|}$

**Coverage (C)** counts how many solutions from the set of non-dominated solutions $D$ are not dominated by the reference Pareto front $R$ [90]. The coverage $C$ is normalised and the larger its value the better.

$$ C = \frac{|\{s \in A \mid \nexists s' \in R, s' > s\}|}{|R|} $$

(2.8)

Following Wang et al. [91] practical guide for selecting quality indicators, these metrics can be split into four categories. Table 2.1 summarises these categories, their aimed goal, and the most common metrics in each of them, in addition to showing the requirements of each of them.

We see that all metrics (with the exception of the number of non-dominated solutions) have their requirements in order to be used. However, not all these requirements are easy to obtain. Given the intractability of the problem I address in this thesis even in the mono-objective case, it is not possible to obtain the ideal point in a reasonable time, thus forbidding me from using the ED metric. The problem is also intractable in multiple objectives, making it hard to obtain the optimal Pareto front or a good enough reference Pareto front that always dominates the Pareto fronts obtained by the algorithms, thus reducing the list of usable metrics to only HV and #Sol.
Table 2.1: Categorisation of the quality indicators with their requirements.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Category</th>
<th>Metric</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>Combination</td>
<td>HV</td>
<td>Reference Point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGD</td>
<td>Reference Pareto front</td>
</tr>
<tr>
<td></td>
<td>Convergence</td>
<td>GD</td>
<td>Reference Pareto front</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ED</td>
<td>Ideal point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\epsilon$</td>
<td>Reference Pareto front</td>
</tr>
<tr>
<td>Diversity</td>
<td>Diversity</td>
<td>#Sol</td>
<td>Nothing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GS</td>
<td>Reference Pareto front</td>
</tr>
<tr>
<td></td>
<td>Coverage</td>
<td>C</td>
<td>Reference Pareto front</td>
</tr>
</tbody>
</table>

2.2 Related Work

In this section I survey the literature relevant to this thesis in both mono-objective VM reassignment and multi-objective VM reassignment for different types of data centres (centralised and distributed). In addition, I survey the different objectives that are the most recurrent. Given the volume of the literature, I only present the most significant works amongst them and redirect the reader to surveys with exhaustive studies in each of the sections whenever it is possible.

2.2.1 VM Reassignment in Centralised Data Centres

The VM reassignment (a.k.a., the machine reassignment) is a problem that aims at reassigning processes/VMs that are already in their initial PM to different ones while respecting a set of hard constraints and reducing a certain cost. The machine reassignment problem (MRP) belongs to the assignment problem family [92].

In the context of centralised data centres, a resource controller has full control over the allocation of any VM to any PM in the data centre. In this context, the MRP has a particularly similar formulation as two theoretical assignments in the literature: the d-dimensional bin-packing (see Section 2.1.2.1).
and the Generalised Assignment Problem (GAP [93]). Both the d-dimensional bin packing and the GAP aim to optimise the cost of assigning a set of jobs to a set of agents, with a job assigned to one and only one agent and also with respect to the constraints of each of the agents. Despite having similar constraints, the former is mostly dealing with the minimisation of the number of used agents whereas the latter can vary from a work to another.

2.2.1.1 Prior to the Google ROADEF/EURO Challenge

In the important and challenging context of managing resources in centralised data centre environments, several optimisation problems have been defined and studied in the literature [94]. Bobroff et al. [50] propose a First Fit algorithm which reassigns VMs in order to reduce the number of PMs running at the same time, while not exceeding the number of negotiated SLA violations. Furthermore, they perform a thorough evaluation of their technique on a realistic VM resource consumption trace. Doddavula et al. [95] also work on optimising resource utilisation by reassigning tasks (or VMs) to different PMs. They evaluate the applicability of d-dimensional bin packing techniques for a fast VM consolidation and compare several variations of the commonly used First Fit algorithm. Beloglazov et al. [9] deal with the reduction of data centres’ energy footprint. They define an architecture and a set of principles for more energy-efficient data centres. They also propose a heuristic based on their architecture that migrates VM in a way that improves the energy efficiency of the data centre, while at the same fulfils the contracted Quality of Services (QoS). Stillwell et al. [96] push the boundaries further by considering a resource allocation with heterogeneous PMs. They show on a simulated experiment based on workload data from Google, that approaches proposed for the d-dimensional bin packing with homogeneous bins can be extended to the heterogeneous case and still provide solutions of good quality. Sharma et al. [97] propose an MILP formulation of the problem and put an emphasis on the elasticity of VMs (migrating a VM while increasing or decreasing the amount of resources it was allocated to meet the SLA of the service it hosts) and design a set of greedy heuristic techniques (called Kingfisher) to optimise it. Zhang et al. [98] aim at minimising the risk of overloading PMs in the future for data centres with intentional over-committed resources (i.e., data centres which provision more resources than what their capacity allows, with
the hope that some VMs will not run or will run with a lower resource demand). They design a technique which controls and minimises the number of migrations of VMs with large resource utilisation.

The solutions described upwards are all based on a centralised optimisation and are expected to only scale to systems with hundreds of thousands of servers. For scalability reasons, some works investigate the hierarchical organisation of the resource controller within the centralised data centre. Jung et al. [99] propose a holistic controller framework with a scalable algorithm (called Mistral) able to handle large scale infrastructures through a multi-level adaptation of the resource controller hierarchy. Zhu et al. [100] design an architecture called ‘1000 islands’ which consists of three different controllers each of them with its time complexity and objective. 1000 islands is both scalable and fast to find a suitable VM reassignment, making it ideal when trying to reduce service level violations of VMs with high priority.

Other works propose a distributed organisation of the resource controller within the centralised data centre. They push the scalability boundary further (the processing is distributed over different PMs and can scale-out) and increase the robustness and tolerance of the system to faults (having other resource controllers which can take over the management if any of them fails). Feller et al. [101] propose Snooze; a VM management framework that is scalable, self-organising (the system configures and sets a hierarchical architecture on its own), and self-healing (equipped with fault tolerance at each of its levels). They also evaluate their framework in a real environment that comprises 144 PMs. Wuhib et al. [102] design and implement a distributed resource management system that builds on top of OpenStack (an open source software for creating and managing private data centres and public clouds), which monitors relevant metrics in real time. Their system supports a large and extensible set of objectives to perform its resource reassignment and switches online between different objectives. They demonstrate their system using load-balancing and energy efficiency management objectives and show how they can update the controllers so that the resource reallocation optimises them.

The majority of these works deal with slightly different formulations of the VM reassignment problem [103] (e.g., different objectives and models for network communication, or allowing constraint violations, etc.). Without an accepted general formulation for the VM reassignment, many works provide
2.2 Related Work

their own definition of the problem and finely tuned algorithms to address it. This negatively impacts the VM consolidation field on many levels [103]:

- It restricts the reproducibility of the results with sometimes papers failing to formally define the problem they study and where the reader has to figure out the problem from the discussion sections or indirectly from the algorithms that are put forward.

- It reduces the applicability of algorithms as they constantly need to be modified to fit the new problem and even making them inapplicable if they combine heuristics or rules specific to the initial problem but that do not apply to the new ones.

- It renders impossible to perform a fair comparison between different algorithms that make the state-of-the-art.

2.2.1.2 Google ROADEF/EURO Challenge

Due to the increasing popularity and scale of the data centres, Google (one of the leaders in the market) proposed a challenge at the ROADEF/EURO [19] forum with a detailed formulation for the Machine Reassignment Problem, where a set of processes (VMs) that are already sitting on a set of PMs have to be reassigned within this set of PMs while optimising a weighted sum of different objectives (i.e., load-balancing of resources and migration costs) and respecting capacity (e.g., CPU and RAM) and other operational constraints (e.g., dependencies and conflicts).

The ROADEF/EURO contest was first organised in 1999 and appears regularly since then (often on a two-year basis). This challenge is organised mutually by ROADEF; the French Operational Research and Decision Aid society and EURO; the European Operational Research society. It also always addresses the optimisation of an industrial problem that is proposed in collaboration with an industrial partner. The 2016 edition was in collaboration with Air Liquide (i.e., a world leader in the supply of industrial gases) and was dedicated to the inventory routing problem. Whereas the 2014 edition was jointly organised with SNCF (a French railway company) and ported on the management rolling stock units.
2.2 Related Work

The edition that is the closest to my topic; the 2012 was in collaboration with Google and was dedicated to the machine reassignment problem. The challenge attracted a large number of participants [104] with 82 registered teams from both industry and academia, coming from several countries (see Figure 2.2). 48 teams sent a program for the qualification phase which consisted of 10 ‘A’ (small and medium scale) instances. 30 teams amongst them qualified to the final round and got offered a new dataset consisting of 10 ‘B’ (large scale) instances to train and refine the tuning of their algorithms. 27 teams sent their program to the final round and their algorithms were evaluated on the ‘X’ (not disclosed) instances. Five financial awards were given to the best teams in the 25th European Conference on Operational Research which was organised in July 2012 in Vilnius, Lithuania. Both best open source submission of €10,000 and best senior submission of €5,000 were awarded to Gavranović et al. [105] (i.e., team S41). Three best junior submissions of €3,000, €1,500 and €500 were awarded respectively to Jaśkowski et al. [106] (i.e., team J12), Brandt et al. [107] (i.e., team J25) and Sansottera et al. [108] (i.e., team J33).

Figure 2.2: Distribution of teams taking part in the Google ROADEF/EURO 2012 Challenge. <Source: “http://challenge.roadef.org/2012/files/Roadef - results.pdf”>
2.2 Related Work

2.2.1.3 Complexity of the Problem

It is indicated by Portal et al. [109] that the machine reassignment problem that was proposed in the ROADEF/EURO challenge is an NP-Hard problem. This can be concluded based on its composition of several NP-Complete sub-problems such as the bin-packing or the d-dimensional bin packing.

To grasp what that means in terms of practicality, Portal and Ritt [110] modelled the problem as an MILP program and evaluated in their Bachelor thesis report the performance of a commercial MILP solver (i.e., IBM ILOG CPLEX). They showed that CPLEX could only solve four small instances within one hour of execution time. They also showed that even when given a longer execution time (i.e., 10 hours), CPLEX struggles to solve all the instances and only solves eight small and medium ones.

2.2.1.4 Algorithms Submitted to the Challenge

The ROADEF/EURO challenge 2012 has not only attracted many participants, but has also seen algorithms of different types. These algorithms have been evaluated and categorised based on the method they employ by Afsar et al. [104] and the best performing amongst them have been compared by Portal et al. [109]. I survey in this section the best performing algorithms that were proposed the ROADEF/EURO challenge. I also report algorithms that stand out with the type of techniques they use in comparison to the majority of the submissions.

Local Search Based Algorithms are the majority of the algorithms that were proposed to the Google ROADEF/EURO challenge. They are based on one of the multiple variations of the local search algorithm. They are globally the best performing techniques as the best approach [105] is based on a local search algorithm and also the second best approach [111] is a hybrid algorithm which exploits a local search technique (see below for more details). Gavranović et al. [105] won the challenge with a Variable Neighbourhood Search (VNS). They define two lower bounds to evaluate the performance of the local search part of the VNS. Their neighbourhoods are generated by selecting a set of promising processes to be part of the rearrangement and applying different
2.2 Related Work

moves on each of them to generate neighbour solutions. Their VNS applies two or four different type of moves at every iteration: two that are simple (i.e., a shift of the assignments and a swap of assignments between processes) and two that are large and time consuming (i.e., a big rearrangement of processes and a ‘chain shift’ of the assignments). Pécot [112] use a Tabu search with two types of moves (i.e., switch of assignments and insertion of a process in another PM). However, instead of exploring the whole neighbourhood of a solution and in order to reduce the time required for an iteration of the local search, they only explore a randomly selected subset of the neighbourhood. Masson et al. [113] use a Multi-Start Local Search with a shift and a swap of the assignments as moves to generate the neighbourhoods. Similarly to the algorithm proposed by Pécot [112], they also restrict the size of the neighbourhood by only randomly selecting a subset of it. Wang et al. [114] submitted a Multi-Neighborhood Local Search. Their algorithm uses three primary neighbourhood structures that aim at efficiently explore the search space. It also uses an auxiliary neighbourhood structure as a perturbation operator to avoid being stuck in local optima.

Simulated Annealing Based Algorithms were also submitted by a few teams. Their solutions were also among the best overall. Portal et al. [109] propose a Simulated Annealing with neighbourhoods selected randomly using either a swap of the assignments of two processes or a reassignment of a process. They also put forward a data structure that allows the evaluation of the neighbourhood in a constant time. Sansottera et al. [108] also propose a Simulated Annealing algorithm that they run in parallel with a Variable Neighbourhood Search. Both techniques use the same moves to generate their neighbourhoods (i.e., a swap between assignments of two processes and a shift of the assignments). Both techniques communicate with each other through a shared solution pool made of two sets: (i) the first containing highly optimised solutions, and the second containing diverse ones. A path relinking is also performed on randomly selected solutions (i.e., one from each of the sets) to explore their intermediary solutions.

Greedy Based Algorithms were also submitted by a few teams to the Google ROADEF/EURO challenge. Most of them have a good performance on small and medium instances, but struggle when facing large ones. Gabay et al.
2.2 Related Work

[115] is the only team to design an algorithm based on a Greedy Randomized Adaptive Search (GRASP). Their algorithm starts by performing a First Fit Decreasing procedure to construct the initial solution, followed then by a local search that uses a swap and a reassignment of processes as moves to generate its neighbourhood. Vancroonenburg and Wauters [116] propose a Late Acceptance Hill Climbing technique that only accepts to move into a solution if it is better than the best solution of the last ‘n’ iterations (n being a constant they define in their algorithm). They use two neighbourhood moves (i.e., a process reassignment and a swap of the assignments of two processes). They also use two different evaluation functions (i.e., the first is more precise but takes time to compute and the second is simpler but quick to compute). Gogos et al. [117] also propose a Late Acceptance metaheuristic that they combine with a Variable Neighbourhood Search to work in cooperation for the duration of the execution.

Hybrid Algorithms were also among the most frequently submitted techniques, with either a combination of multiple metaheuristics together or metaheuristics (metaheuristics combined with mathematical techniques such as MILP and CP solvers). These techniques were also performing well as the best junior submission [106] and second best approach [111] are hybrid algorithms. Jaśkowski et al. [106] (the best junior submission) combine three phases in their method. They start with a greedy Hill Climbing that looks for a good initial solution. They follow then by a set of finely tuned heuristics. They finally run a search that iteratively uses dynamic programming to select a subset of PMs and processes and reassign them optimally using an MILP solver. Mehta et al. [111] achieve the second best results overall using a Large Neighbourhood Search that they combine with a Constraint Programming solver to evaluate the neighbourhood at every iteration and select the best solution to move to. Given the complexity of running the Constraint Programming solver on the whole neighbourhood, the authors only select a subset of processes and PMs to take part in the rearrangement. Brandt et al. [107] also combine a Large Neighbourhood Search with Constraint Programming techniques. Unlike Mehta et al. [111], they only select a subset of processes (not PMs) that they feed to a CP solver to find their best reassignment. They also take advantage of multi-threading in their optimisation by running several local search algorithms at once and synchronising their best solutions at the end of every
iteration. Teypaz [118] also integrate an exact algorithm with a metaheuristic. They use a two-phase Tabu search with two elementary moves (i.e., process reassignment and swap between assignments of two processes), and generate an intermediary problem (i.e., a maximum weight matching problem) that they then solve to optimality using a blossom shrinking technique in order to select the best move. Lopes et al. [119] use four different iterated local search heuristics with two versions relying on Integer Programming based perturbations and the two others relying on random perturbations. Gharbi et al. [120] put forward an MILP model that they solve using a linear solver. However, they could only tackle small scale instances and could not address efficiently larger ones.

2.2.1.5 After the Challenge

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. [121] improve the work of Mehta et al. [111] 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. [122] 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. Their aim is to use different heuristics to deal with various neighbourhoods, that are selected on-line according to their pertinence on each instance. The first level contains algorithms that deal with large neighbourhoods and take a long time to compute, thus only used when the current solution is of a far worse quality than its neighbours (it is worth exploring the neighbourhood). Whereas the second level uses smaller neighbourhoods, it is less time consuming and it is used when solutions of the neighbourhood are of a similar fitness as the current one (it is not worth spending more time for the improvement the neighbourhood brings).

Recently, Turky et al. [123] proposed an evolutionary parallel Late Acceptance Hill Climbing algorithm. They generate a population of different initial solutions and run a Late Acceptance Hill Climbing algorithm on each of
them in parallel. Having different starting solutions and search processes with different paths increases the chances of avoiding a same local optimum and eventually allowing one search process to reach the global optimum. They also show in an experimental study that their method outperforms its sequential counterpart and achieves comparable or slightly better results than the state-of-the-art. The same group of authors develop a year later an Evolutionary Simulated Annealing (ESA) algorithm \[124\] by taking out the Late Acceptance Hill Climbing algorithm and replacing it with a Simulated Annealing. Unlike a regular SA that uses one single solution, ESA uses a population of solutions, with an SA for each of the solutions that runs in parallel. Additionally, they use a mutation operator to perturb the search of an SA in case of convergence to a local optimum, and have all the parallel processes working for the whole duration of the experiment. They have also showed that their algorithm can outperform the state-of-the-art on some instances. In another work from Turky et al. \[125\], the authors do not aim at finding the best solution but rather 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. A comparison of two neighbourhood structures with the Steepest Descent shows that using a combined neighbourhood structure allows achieving better results than using a single neighbourhood.

### 2.2.2 VM Reassignment in Distributed Data Centres

The constant evolution in the organisation of large data centres, their diverse feature offerings and their difference in cost and availability have led to a progression from in-silo (centralised) data centres to distributed ones. We have witnessed in the recent years (and we continue to witness) the emergence and the rise of different types of distributed data centres ranging from multiple clouds (between data centres of different cloud providers) to decentralised data centres (within autonomous departments of the Same company or cloud provider), to hybrid clouds (a mix between private data centres and public clouds).

Driven by reductions in operation costs, high service availability and the need for clients to access special features, we witnessed in the recent years
2.2 Related Work

the emergence of multiple clouds. Therefore, cloud providers are more numerous to exploit resources from other cloud counterparts and to put their own available resources at the disposal of others to reduce their amount of resources sitting idle. In multiple clouds, given that cloud providers often correspond to different companies, they worry mostly about the good execution of their services (sitting on top of VMs) and their pricing, when using the infrastructure of other cloud providers. Despite the numerous names behind the delivery models of multiple clouds (e.g., cloud of clouds, aggregated clouds, cloud merge, etc.), there are two main delivery models that enable multiple clouds [16] and they depend on the existence or not of a prior agreement between the different cloud providers: multi-clouds and federations of clouds. There is also a third delivery model (i.e., inter-clouds), but this one is nothing more than a combination of the aforementioned ones and applies their same principals and VM allocation techniques, so I do not include it in this related work.

2.2.2.1 Federations of Clouds

In this delivery model, cloud providers have a prior agreement between each other in order to set up the federation, therefore increasing the capabilities of the involved Clouds and enlarging the pool of services they offer to their consumers. Federations of clouds are based mainly on IaaS with a requirement of interoperability in the used virtualisation technology between all its counterparts, with Clouds offering to others access to a part of their infrastructure that they can use to run their services. Darzanos et al. [126] propose three different modes under which a federation of clouds can form with an incentive for each of the entities taking part in it. The modes have different degrees of interactions between the Cloud providers (i.e., strong, weak and elastic) to decide on the federation policies.

There are several examples of federations of clouds in place. However, given the hardness of establishing the agreement between commercial cloud providers, this delivery model is mostly famous among research and academic data centres. For instance, the European Grid Infrastructure (EGI) Federated Cloud (FedCloud [127]) gathers 16 private academic Clouds. FedCloud is built using open standards with a focus on fulfilling needs of the scientific community. Diaz et al. [128] report on one of the experiences in setting up a
federation of clouds made up of desktops in different universities in Colombia. Their federation of clouds gathers existing and idle computer resources in the different participating institutions to run High-Performance Applications on them when all the dedicated infrastructures are busy. FELIX [129] is another project that leverages the federation of clouds model with entities called Software Defined Networks (SDN) islands to develop a global scale testbed for the migration of entire IaaS from one continent to another in case of major catastrophes. Takefusa et al. [129] use FELIX to study the migration time for performing such a migration between Europe and Japan, when subject to different latencies and show that a migration can be achieved within a 10-minute period that they judge reasonable in their context.

The establishment of federations of clouds is favoured by the development and multiplication of enabling technologies. The European Organisation for Nuclear Research (CERN) created a project called Openlab [130] with the aim of accelerating the development of seamless federations among several public and private data centres that are built on the OpenStack platform. Arjuna [131] designed a platform for a dynamic formation and adaptation of a federation of clouds. Their platform is made of resources that are made available by independent cloud providers under certain policies. To ease the contribution of resources and the consolidation of VMs, Arjuna also developed the Agility framework which allows organisations to share their resources to consolidate their services.

More recent systems try to leverage the resource elasticity that offer federations of clouds for the deployment of MapReduce-based applications. For instance, Panarello et al. [132] design a federated system that adds a shared memory system located on the Amazon S3 Cloud Storage Provider to reduce the overhead led by the communication and transmission of data between the different Clouds. Villari [133] propose to apply such a model of federation of clouds for MapReduce-based video transcoding to cope with large and constantly increasing number of shared video-selfies.

There also many approaches that aim at optimising resource utilisation in federations of clouds. These approaches are most of the time embedded in the framework that enables the establishment and management of the federations. Some of these strategies have been surveyed by Gahlawat and Sharma [134] and can be split into two categories: (i) centralised and (ii) distributed.
Many of the approaches use a centralised controller to manage the resource selection, the booking of resources and the resource allocation processes. One instance of these centralised approaches is built within the Contrail [135] framework which manages the federated resources with the aim of enforcing and fulfilling the QoS that is defined as SLAs at the federation level. The framework also allows a negotiation of VMs performances before mapping them onto the PM that achieves them. ClouDIA [136] is another framework that uses a centralised approach. ClouDIA assigns and runs VMs in four steps: (i) the client requests one or several instances from ClouDIA with a corresponding required communication graph, (ii) ClouDIA allocates more instances than required in order to maximise the chances of having a successful deployment even with the latency, (iii) ClouDIA uses MILP and CP solvers to search for the best allocation plan in terms of pair-wise latency between nodes, and (iv) ClouDIA terminates any unused instance.

There are also some approaches that use a centralised controller to perform the resource management, but provide the Clouds taking part in the federation with more direct communication channels to negotiate their resource utilisation. RESERVOIR [137] is one of the major frameworks that provides such features. RESERVOIR is a European research initiative [137] which investigates ways cloud providers could lease their idle resource capacity to other providers in punctual need for additional resources. In RESERVOIR, cloud providers interact directly with each other to request and negotiate the resources, with their corresponding SLAs. However, the framework is still responsible for the VM allocation and bases its optimisation on the Column Generation method with the aim of fulfilling all the SLAs.

Unlike the aforementioned approaches, Dupont et al. [138] propose in their work a framework that is independent of the management system, that reassigns VMs in federated data centres with the aim of reducing the power consumption while at the same time fulfilling requirements in terms of performance. Their framework provides a Constraint Programming language for expressing the SLAs as constraints which gives a large flexibility to the users, and looks for the VM reassignment that satisfies best the SLA constraints. The framework also uses a CP solver for the optimising the VM reassignment problems.
A growing trend is moving federations of clouds from centralised approaches to distributed ones and is using distributed controllers to manage the resources in the federation. Lee et al. [139] propose a distributed approach to the VM allocation. Their approach groups VMs based on the communication profiles in order to reduce communication overheads. It also uses a modified 0/1 knapsack to find the best allocation plan with a reduced time complexity. Furthermore, their approach improves resource allocation through a reallocation of VMs and motivates Clouds to federate their resources with evolving pricing strategies that reflect both the resource competition and the marginal costs of over-provisioning. Given the competition for resources between the different Clouds, their system reassigns VMs until reaching an equilibrium where no Cloud can improve its cost with a unilateral VM reassignment or price fluctuation. Hassan et al. [140] propose two game theory based distributed VM allocation approaches for federated clouds. In the first one, Clouds are assumed to be cooperative and in the second one, Clouds are considered non-cooperative. They show in an experimental study on a federation with homogeneous Clouds that in a cooperative scenario, their VM allocation mutually motivates all the Clouds to federate their resources. They also show that in the non-cooperative scenario with a game solved to optimality, Clouds are not guaranteed to achieve a benefit by federating their resources. Carlini et al. [141] propose an approach that is both highly distributed and self-adaptive for the optimisation of the overall VM deployments through point-to-point interactions between heterogeneous Clouds in the same federation. Each of the Clouds is assigned an agent that is devoted to the optimisation of the assignment and reassignment of its VMs. Every agent follows successively five simple tasks going from updating the information of its Cloud, to selecting Cloud counterparts, to initiating the contact and getting its up-to-date status, to analysing the potential of the Cloud counterpart in improving its gain, and to making the VM exchange. Additionally, the authors formally define the VM exchange problem and provide with its equivalent Markov-chain model.

2.2.2.2 Multi-clouds

Systems based on the multi-cloud model are similar to federations of clouds as they allow Clouds to extend their resource capacity by consuming resources
from other Clouds. However, in multi-clouds, a Cloud does not require either a prior agreement with its counterparts or an interconnection and a sharing of their resources. Instead, the resource acquisition is delegated to third-party companies or middlewares which build unique entry points to the public clouds by leveraging their default IaaS interfaces [142]. This model is more attractive to industry as (i) it brings added value for both the third-party companies which put this service in place, (ii) it allows data centres to quickly and easily extend their capacity, and (iii) it is not intrusive to the Cloud providers. Additionally, unlike federations of clouds which required an interoperability between the different participating Clouds, multi-clouds only need a portability tool that ensures the translation of the virtualised services from their original hosts to their destination ones.

In the recent years, we witnessed a rise in the industry’s interest for multi-clouds and an increase in the middleware and third-party offerings for such a model with each of them competing to provide a single entry points to as many cloud providers as possible. Some of the middlewares are provided in the form of libraries for developers of new services which consume resources in the Cloud, allowing them to write a code that is Cloud agnostic. The Java multi-cloud toolkit (Jclouds [143]) from Apache is one instance of these libraries. Jclouds allows portability across multiple cloud providers for Java Virtual Machine (JVM) based services. δ-Cloud is another library written in Ruby and based on an RESTful Application programming interface (API), which allows connecting to several public cloud providers such as Amazon EC2 and GoGrid, and private data centres built using different frameworks including OpenStack and Eucalyptus. Other multi-clouds enabling middlewares are the result of open source projects and are provided as deployables. The most know amongst them is mOSAIC [144] which offers an open source API with a deployable control of VMs life cycle.

In parallel to deployable middlewares, many third-party companies have created their own multi-cloud tools that they host on-line and provide as a service to their costumers. The most known of these hosted middlewares are offered by RightScale [145] and Kaavo. RightScale [145] provide an on-line platform for managing and deploying cloud resources either in public or private environments, and provide their clients with a controlled and automated deployments of their VMs and enables a fine-tuned configuration of their deployment. RightScale includes a variety of public clouds such Amazon Web
2.2 Related Work

There are several factors that led to the multiplication of works on the VM consolidation in multi-clouds going from the increase in interest from industry, to the important saving that they enable. The studies put forward several techniques to optimise different parts of the multi-clouds. The works that are the most related to mine describe the problem as the ‘broker VM (re-) allocation problem’ where the third-party that provides the multi-cloud tries to reassign VMs in its charge between different public cloud providers in order to ensure the satisfaction of SLAs (or at least reduce their violation) and optimise their revenue by leveraging least costly providers at any given time. Nesmachnow et al. [146] provide efficient heuristics which help brokers improve their profit and maximise their earnings by leveraging the important price differences between the cloud providers. They propose a total of seven heuristics and one local search. They compare their algorithms on 400 instances of the problem with realistic scenarios and workloads. Guzek et al. [147] survey different evolutionary algorithms and bio-inspired techniques that are used for the brokering problem and compare them according to the type of workload they handle and the objectives they aim at optimising. Legillon et al. [148] propose a genetic algorithm with several problem specific operators for mutation and crossovers, with the aim to optimise the cost of the overall VMs allocation. They evaluate their algorithm against an MILP solver on a real-life instance of the problem and also on four synthetic instances with different sizes created based on the real-life one. They show that their algorithm achieves good results and in a reasonable time on both small and medium instances, and it achieves acceptable results on large instances when the MILP solver could not find any solution. Somasundaram and Govindarajan [149] propose a Particle Swarm Optimisation based VM allocation mechanism which aims at efficiently managing the brokered resources for a reduction in operations cost and also completing the tasks within times that were specified by the user. They show on a simulated High-Performance Computing (HPC) multi-clouds the effectiveness of their approach in minimising tasks completion time and thus meeting users’ deadlines and increasing their satisfaction. They also
showed on an experimented based on the Eucalyptus environment, that using their technique leads to a noticeable improvement in performance. Iturriaga et al. [150] design a parallel and hybrid evolutionary algorithm for the optimisation of the VM brokering problem with the aim of increasing the broker’s revenue. Their approach combines a genetic algorithm with a distributed sub-populations model and a Simulated Annealing operator. They base their experiments on a set of instances of realistic workloads and real scenarios obtained from cloud providers. They show that their technique outperforms a greedy approach and allows an increase of more than 130% in profit for the broker, while meeting the deadlines in the execution of tasks. Ismail and Cardellini [151] push the boundaries of brokering even further by studying the interactions between different multi-clouds (i.e., brokers) with the uncertainty issue that is led by the adaptation, which may have a negative impact on the VM consolidation. To address their problem, they propose an autonomic and distributed brokering framework, and formulate the VM reassignment as a reinforcement learning problem that they then solve using the Q-learning algorithm.

### 2.2.2.3 Decentralised Data Centres

Decentralised data centres often appear in companies that have large data centres with an infrastructure spread over different autonomous departments (or entities). Given that the different departments are part of the same company, the company worries about both the good execution of its VMs and the performance of its departments (as they affect the global performance of the company). I have previously described such decentralised data centres [152] and proposed a formulation for the VM reassignment problem in that context in one of my works [5].

The decentralisation of resource allocation is fairly covered in the area of grid computing. However, unlike virtualised data centres, grid computing deals mostly with the scheduling of tasks and do not consider the notion of reassignment (or migration). Krauter et al. [153] survey resource management systems in distributed grid computing. They also propose a comprehensive model and a taxonomy for its different architectures. Grid computing organisations are organised into three sets: (i) Flat: where all the PMs can communication with each other and negotiate the resources directly, (ii) Hierarchical:
where PMs have to be in the same level of the hierarchy to communicate and can also communicate directly to the PMs above or below them, and (iii) Cells: where PMs are clustered into cells with PMs able to communicate with other PMs within the same cell as if they were in a flat system, and PMs at the extreme edges of the cells act as representatives when communicating with their outside world. The cell model is the closest to what I call decentralised data centres in this thesis. More particularly, the model where there is only one resource allocator per cell and where each of these allocators is linked to the level above (the global manager of the data centre) in a hierarchical fashion (cells do not interact directly with each other). Several systems have been created to manage resource allocations in hierarchical cell grid computing systems (e.g., Globus, MOL and Nimrod/G). In particular, the work of Quezada-Pina et al. [154] defines a two-level hierarchical cell grid system. However, it is concerned with the scheduling of jobs rather than their (re-)allocation. The work of Dorronsoro et al. [155] also addresses the scheduling of jobs, but in a multi-core distributed system based on a similar two-level hierarchical cells system.

When it comes to the context of data centres, there are several papers in the literature that work on VM reassignment in geographically distributed DCs [156], without considering the autonomy of the entities that make up the data centre when placing the VMs in their infrastructure. For instance, Mehta et al. [157] model the energy cost of geographically distributed data centres with prices varying from one location to another. The Google ROAD-EF/EURO challenge [19] is not different to that as it also concerned by the reassignment of VMs in data centres that combine different locations and enforcing some of the VM replicates (a service duplicated into several VMs) to spread on a certain number of locations for resiliency and reliability purposes. Forestiero et al. [158] propose to organise the geographically distant data centre into clusters with an elected PM on top of each of them to communicate and negotiate the VM re-allocations with the aim of balancing the load. However, the elected PMs in each of the locations interact directly between each other in a peer to peer basis and aim at optimising the global objective of the data centre and not for their local preferences.

The literature that is related to decentralisation is often proposing decentralised solutions for the reassignment of VMs in centralised data centres (see Section 2.2.1) and do not address the case where the infrastructure is
2.2 Related Work

decentralised. This often means that instead of having one resource allocation controller, they spread the decision making by installing sub-controllers throughout the infrastructure to gain further in scalability. However, these sub-controllers are often working with the same objective and with the aim of improving the global data centre, unlike in decentralised data centres where the entities have different preferences between them, which might even be different from the global managers. The framework Snooze that was proposed by Feller et al. [101] is one of these systems that creates sub-controllers for managing VMs and organise the communication between them in a hierarchical format. Another similar system was designed and built on top of OpenStack by Wuhib et al. [102]. In addition to automatically creating sub-controllers, their system allows an on-line adaptation of the global objective and takes care of updating autonomously the objectives at the sub-controller level. Loreti [159] also propose in their thesis a decentralised model for single in-silo data centres and propose an autonomic VM allocation system, where the movement of different VMs across the PMs is made difficult by their heterogeneity. They then move away from the centralised data centres and investigate distributed ones. However, they only address the multi-cloud model and propose an autonomic VM management system for that scenario, while addressing the higher latency medium between the multiple clouds. To validate their algorithm, they take the role of a broker/client who wants to run a data-intensive application that uses MapReduce and leverages the multiple clouds for that purpose.

Similar to the work of Loreti [159] on the decentralised VM management in the context of multi-clouds, there are also other works that study the problem in multiple clouds in their two different models. For instance, Ismail and Cardellini [151] propose an architectural model that allows a decentralised self-adaptation in resource allocation in multi-cloud models. Carlini et al. [141] also propose a highly decentralised and self-adaptive algorithm that tries to optimise VM allocations. However, their work focuses on federations of clouds with a sub-controller installed in every Cloud of the federation in order to manage the point-to-point interactions with their counterparts within the federation. In both federations of clouds and multi-clouds, the goal of every Cloud (in federations of clouds) or brokers/clients (in multi-clouds) is mostly to maximise their profit while guaranteeing that their VMs (the ones they decommission to other Clouds) are performing at their regular level, and
do not consider the situation of the Cloud where they are hosted. Whereas in an Enterprise with decentralised data centres, the objectives that managers value (including profit or energy consumption, but not limited to them) are directly related to the performance of each of the entities that compose their infrastructure. Therefore, the managers are not only concerned with the monetary profit that can be made through VM reassignment, but also with the performance of its different entities on other objectives.

There are also some works that consider the decentralisation of data centres as agents sitting on top of each physical resource (mostly servers and routers) that allow them to interact with each other in a peer to peer basis to optimise their respective utility function. Dubois et al. [160] developed such an approach and proposed a self-organising system called MyCoload that aims at balancing the load. They use a previously defined peer to peer system called MyConet. MyConet is based on super peers and uses a bio-inspired topology to orchestrate the communication and optimisation between peers. In the peer to peer topology of decentralised data centres, the decision making is done at very low levels as decisions of whether/where to reallocate the VM/workload are taken at servers or routers level, with a limited view of the global infrastructure and with sometimes a limited computing power which limit the optimisation to only simple rules or heuristics.

Chen et al. [161] consider a geographically distant data centre, with entities responsible for the VM allocation within their infrastructure. They also consider that all these sites are connected through dedicated/leased network channels where they can communicate and exchange their respective VMs so that they could optimise their operations cost. They also developed a model for an optimal VM allocation in their infrastructure. This work has a similar definition of decentralised data centres to the one I consider in my work. However, theirs differs from mine in two aspects: (i) the sites have the same function (operations cost) that they try to optimise and do not have different preferences, and (ii) the sites negotiate directly with each other instead of having a manager with a better and a more global view on the data centre.
2.2 Related Work

2.2.2.4 Hybrid Data Centres

Hybrid data centres or hybrid clouds is a new model which allows companies to host some of their VMs in public clouds while keeping the most sensitive amongst them (e.g., for security or reliability reasons) in their own private infrastructure.

Examining the available literature in hybrid clouds, we see that the majority of these studies focus on architecture/management aspects of such environments and their challenges [15]. Studies that are concerned with resource allocation are most of the time either (i) focused on the selection of the public cloud, or (ii) focused on the private DC with the possibility of ‘bursting’ to public clouds [162] to cope with a momentary excess of demand in resources.

Both multiple clouds delivery models can be seen as ways of setting up a hybrid data centre with the focus on choosing a suitable public cloud. For example, the multi-cloud service provided by RightScale [145] allows data centres to run their VMs locally and leverage the available set of Clouds in case they want to decommission their workload. Federations of clouds also allow Clouds to keep their VMs in their infrastructure and use computation resources from Clouds of the same federation when needed. However, in federations of clouds, every Cloud plays the roles of both the private and public cloud at the same time: (i) private when they are using the infrastructure of other Clouds and (ii) public when their infrastructure is the one that is being used by others.

Data centres burst to public clouds when they require more resources than what their infrastructure provides or when they necessitate specific features that they do not own. This is mainly due to the fact that bursting is time limited and is only used to cope with the unexpected demand [16]. Given this limited frequency in bursting, the major part of these data centres, focus on optimising the resource allocation in their infrastructure. They sometimes even intentional over-commit resources [98] as they can handle the excess in demand through bursting. In this model, data centres focus mostly on the optimisation of their infrastructure.

Works that deal with VM allocation in hybrid data centres are often driven by overall cost reduction [11], SLA satisfaction [162] or minimisation of VM
2.2 Related Work

leasing cost [163]). Unuvar et al. [164] consider a large geographically distributed private data centre. They study the problem of either assigning jobs in-premise to one of the private data centre locations or to decommission it to a public cloud. They propose an extended Biased Importance Sampling (eBSA) algorithm that is similar to a genetic algorithm but uses the Cross-Entropy [165] measure to generate new individuals based on their distribution. Abbes et al. [166] address the same problem as Unuvar et al. [164] and also aim at minimising the overall cost (including communication cost). They develop a genetic algorithm to optimise it. They then compare its performance against an exact linear solver (i.e., IBM ILOG CPLEX) and show that it achieves near optimal results most of the time. However, despite basing their paper on the work of Unuvar et al. [164], they did not compare against their eBSA algorithm.

There are several other works on VM consolidation in hybrid-clouds but they either (i) address a small scale scheduling problem with a simplistic private DC architecture: Chunlin and LaYuan [167] define a set of routines for an optimal resolution of the problem, while Bittencourt et al. [163] survey common scheduling algorithms and study their performance on the problem, or (ii) put an emphasis on the private DC sub-problem [168]. However, there are a few that take the complex architecture [164] and workload [169] of large DCs into account, but not as decentralised DCs though.

2.2.3 Multi-objective VM Reassignment

Data centres, with several VMs and PMs have many objectives that they consider when optimising their infrastructure through VM consolidation that are often antagonistic. The multi-objective consolidation of VMs has been described recently by Beloglazov et al. [9] as an important research challenge for data centres. I present below the most commonly used objectives, followed by the major works in both centralised and distributed data centres.

2.2.3.1 Commonly Used Objectives

Lopez-Pires and Baran [170] survey extensively the literature and works that are dealing with VM allocation that were published by four major editors
2.2 Related Work

including IEEE, ACM, Elsevier and Springer from 2008 to 2015. They show that energy consumption is the most frequently used, as it was considered with 50.0% of the papers taking it as an objective to optimise. They also show that network traffic and migration in their different variations arrive second and account for up to 30.9% of the total number of papers aiming at optimising it. Maximising the economic revenues of a data centre is taking the third position with an appearance of 22.6% of the time in total. The fourth in their ranking is the maximisation of the data centre performance which includes maximising the QoS, reducing VM interference, satisfying SLAs, etc. that I gather in my work under the hood of reliability. Note that the aforementioned percentages do not add up to 100% as some works take into account more than one objective. Also, given that maximising the economic revenue of a data centre is mostly trying to find a monetary value for each of the objectives they consider (when it is possible) and aggregate them to a single value, that they do not take into account the incompatibilities they might have, and do not consider what these objectives might mean with regards to non-monetary aspects, I decided not to include it in this work.

Reducing Energy Footprint has become a key issue for large data centres, as in addition to its environmental impact, it also has economic and marketing benefits. However, reducing the energy cost is a hard task as it requires a less accurate evaluation of the amount of energy a data centre uses at a particular time, especially given the variety of PM hardware, the difference in cooling techniques from a data centre to another, and also the variation in electricity pricing schemes from a geographical location to another. As it was said earlier, this objective is the subject of several works. A survey of energy aware VM allocation is presented by Beloglazov et al. [9]. They identified these five following open challenges when aiming for an energy aware management of data centre: (i) designing and developing an energy efficient technique which quickly optimises the problem and able to consider multiple resources and several constraints, (ii) taking into account virtual network topologies in order to lower the network traffic and thus reducing the energy consumption, (iii) developing new thermal cooling management techniques to reduce the energy consumption, (iv) developing algorithms able to handle VMs of different workloads, and (v) providing scalable algorithms through their decentralisation.
2.2 Related Work

It is complex to model the electricity consumption of a PM. Most works in the literature use a simplified model that is defined in the paper of Lien et al. [171], whereby the electricity consumption can be represented as a linear function based on the total usage in CPU. De Maio et al. [46] extend the definition of energy consumption in their VM reassignment problem by modelling and taking into account the energy consumption of network transfers and virtual machine migration. Mehta et al. [157] model the energy cost of geographically distributed DCs under varying demands and prices according to the location and time of the day. They propose a CP approach that they combine with LNS for the reassignment of all the VMs while taking into account the migration and operation costs, in order to lower the energy cost of the infrastructure.

Optimising Network Traffic and Migration Cost are also the concern of several papers. This is the objective with the most diverse formulations as network topologies differ from one paper to another and there exist several network related metrics on which you can compare VM (re-)allocations. Lopez-Pires and Baran [170] mention 18 objectives that are related to network traffic and VM migration. This list includes minimisation of network traffic, average latency, number of migrations, migration time, etc.

It appeared from Lopez-Pires and Baran [170] survey that minimising network traffic is the most frequently addressed in the literature when considering this category. Most of these works address the VM (re-)assignment in the context of data centres with service oriented VMs (e.g., public cloud providers) or HPC clusters. In these contexts, VMs are highly communicating with each other and minimising the network traffic, latency, and end-to-end response time seems to be more than appropriate. Google introduced in the ROADEF/EURO challenge 2012 [19] the concept of neighbourhood where VMs that are highly communicating with each other or requiring fast connections between them have to be assigned to PMs within the same neighbourhood (a set of PMs that are either part of the same rack or connected in a way that fulfils these connection requirements). This move from Google is a long leap as improving the networking is not considered as an objective anymore, but rather as a hard constraint that needs to be fulfilled.

It also appeared from Lopez-Pires and Baran [170] survey that many works try to reduce the number or the duration of VM migrations or try to mitigate
the impact of these migrations with less accuracy. Pasumarthy [47] investigate in their work the complex process of live migration and measure some of its ramifications. In the Google ROADEF/EURO challenge, they included the migration as one of the objectives to minimise and take into account both the time needed to prepare the VM for migration, the time it needs to be deployed and also the duration of the PM to PM migration. In my work, I also used this cost as one of the objectives to optimise.

**Improving Reliability** is also an important factor for data centres when managing the resource allocation, especially for data centres running VMs with critical workloads that have to be working and at the required performance level at all times.

Resource contentions (e.g., cache) can be generated when hosting multiple VMs on the same physical machine leading at best to a degradation in their performance (thus violating some SLAs) or at worst to a failure of the PM. The reliability takes a different formulation from one paper to another and it often takes the form of reduction in SLA violation when these SLAs are not expressed as hard constraints (constraints we cannot violate). Gandhi et al. [172] analyse historical traces of VM workloads in order to identify some usage patterns according to the time and the day (e.g., long-term workloads, peak hours, etc.). They combine in their VM management both prediction of the VM resource demand and reaction towards changes in workload. They show in a study on a realistic workload trace that their approach reduces significantly the amount of violations in SLAs when compared against other techniques. Garg et al. [173] also aim at reducing the number of SLA violations, but they target data centres with VMs that have heterogeneous resource demands and usage patterns (different types of application). They propose a VM admission control mechanism and show in an experimental study that it leads to a reduction in SLA violations over a static VM assignment.

VM reassignment algorithms that do not take the reliability in consideration are often assumed to buffer the reserved resources as safety capacity [174], hence allowing them to not consider the potential issues of VMs co-location. Recently, it has been proposed to adapt the size of the buffers based on the workload of VMs [8] and their resource utilisation patterns. Therefore, trying to assign VMs with similar resource needs at any given time to different
2.2 Related Work

hosts, reducing the risk of violating resource requirements. Similarly, Google introduced the concept of safety capacity in the ROADEF/EURO challenge [19] at which a PM may not be able to satisfy its SLAs. This definition is used by the numerous works (described in Section 2.2.1) which propose a technique to address the machine reassignment problem. This is also the definition I consider in my work.

Other Objectives There are also several other objectives that are used in the literature with a lesser occurrence [170]. These objectives include but are not limited to the maximisation of resource utilisation, the reduction in the number of used PMs, and the reduction in resource wastage.

2.2.3.2 Centralised Data Centres

There are several works in the literature that try to optimise centralised data centres trough the (re-) assignment of VMs, while considering several objectives at once [175–177]. Pires and Barán [175] recently surveyed some of the works in the VM (re-) allocation field and found that 36% of them are taking two or more objectives in consideration when performing their optimisation, which indicates a clear interest in considering multiple objectives and stresses the importance of not only focusing on one of them. These works differ from each other in the way they consider the objectives as some works favour one over the others, combine them into a single objective, or keep them independent.

Complexity of the multi-objective VM reassignment can be implied from the complexity of its single objective counterpart. It is indicated in the work of Ehrgott [178] that: “a multi-objective counterpart of an NP-Hard single objective problem is also NP-Hard” (p1).

In my case, I have:

- The multi-objective VM reassignment in centralised data centres is the multi-objective counterpart of the machine/VM reassignment problem in centralised data centres (p2).
• The machine/VM reassignment problem in centralised data centres is NP-Hard (see Section 2.2.1) \((p3)\).

Therefore, based on \((p1)\), we can say that \((p2)\) and \((p3)\) imply that: The multi-objective VM reassignment in centralised data centres is **NP-Hard** as well.

**Favouring a Single Objective** is one way of dealing with multiple objectives. There are some studies in the literature that model the problem of VM consolidation with multiple objectives before putting an order of preference for them. This order is then used to favour one of the objectives in the optimisation.

For instance, some works deal with the multiple objectives by ‘downgrading’ some of them to the level of constraints and only keeping one of them as an objective to optimise. The objectives that are turned into constraints are forced to satisfy a pre-defined performance (lower thresholds), but they are not optimised beyond these thresholds. Furthermore, the definition of these thresholds is critical as in the minimisation case, small thresholds lead to a risk of unsatisfactory problems (no existing feasible solution) and large thresholds imply bad performance on the given objectives. For instance, Sato et al. \([179]\) reassign VMs according to a predictive model of their demand on resources. They aim to reduce the energy consumption of the PMs while maintaining a good quality of service in a data centre with over-committed resources. However, instead of considering both objectives as such, they turn the quality of service into a constraint based on the amount of CPU and RAM that are remaining in every PM with a defined rate to not exceed.

There are also some other works that put an order on the objectives and optimise them iteratively using their lexicographic order. Chen et al. \([180]\) propose to use a strategy with a dynamic objective with a completion time of the VM tasks as the first objective to optimise, and then change the objective to minimise the operation cost of the data centre.

**Aggregated Objectives** is the most common way of addressing the VM reassignment problem, despite a large number of works dealing with multiple objectives. The majority of them use a scalarisation of some sort to combine the objectives they consider. They then optimise the resulting problem as a single objective one. Pires and Barán \([175]\) estimate in their survey of VM
(re-) allocation approaches that 32% of the studied papers are defining their problem as a multi-objective one, but end up aggregating the objectives and solving a mono-objective one.

There are three common aggregation approaches in the literature [147]. The first and most common approach consists of using a weighted sum of the objectives. The second approach uses an Analytic Hierarchical Process (AHP) to make consecutive trade-offs between the objectives. The third and last approach applies fuzzy logic to combine the conflicting objectives in a more flexible way.

These approaches require a deep knowledge of the problem in order to define a correct combination of the objectives –which is not always possible. In addition, some combination approaches such as the weighted sum only permit to get the non-dominated solutions that are on the convex hull of the Pareto front if these objectives were to be taken separately. Thus, potentially missing a large number of solutions with more granularity in their objective variations and better trade-offs, which would have otherwise made sense to the decision makers.

The linear combination of multiple objectives through a weighted sum is the most used of these aggregation techniques. For instance, Mehta et al. [157] propose an algorithm which combines a CP solver and an LNS metaheuristic for the VM reassignment, with an objective which compromises several other sub-problems through a weighted sum. In their work they optimise the weighted sum of the number of VM migrations, the load of PMs and the energy cost of geographically distant data centres. The Google ROADE-F/EURO challenge [19] does not make the exception and so as all the papers that address its problem [104, 106, 119] (see Section 2.2.1 for an exhaustive list and a description of these works). Google proposed to optimise their machine reassignment problem with an objective function that combines three sub-objectives as a weighted sum. The used sub-objectives are (i) the load cost which corresponds to sum of utilisation on each of the PMs, (ii) the balance in resource utilisation with a target to reach for each PM, and (iii) the migration cost that aggregates three elementary costs i.e., the cost of preparing processes for the migration, the cost of migrating them from a PM to another and the balancing in number of processes migrated in every service.
2.2 Related Work

There are also some other non-linear combinations that consider a different type of distances to an ideal point (an imaginary point that is the best on every objective). For instance, Liu and Jia [181] combine four objectives using a Euclidean distance to the ideal point and then optimise their initial VM placement with this unique objective. They propose an improved Particle Swarm Optimisation technique, which generates the initial particle using a greedy approach and defines a fitness function that they combine with a Bayesian transformation to define a measure of credibility of solutions. They compare their algorithm against two other algorithms according to their final Euclidean distance in a Cloudsim based simulation with up to 500 VMs and a number of PMs that is not limited (but seems to be a maximum of 180 according to the results they obtain).

Few works optimise the objectives trade-off, after generating the most suitable trade-off using the AHP process. For instance, the authors Kord and Haghighi [182] optimise the VM allocation with regards to two objectives: (i) the reduction in the number of SLA violations and the minimisation of the data centre energy consumption. They use AHP to make a trade-off between these two objectives that they then optimise using a Minimum Correlation Coefficient (MCC) method. They also show in a simulation based experiment (based on Cloudsim) that their approach offers a suitable trade-off when optimising both objectives at once.

The rest of the works base their objective combination on fuzzy logic, which allows a more flexible design of levels of preferences. For example, Song et al. [183] consider the optimisation of multiple perspectives in their work: (i) reduction in the number of used PMs, (ii) minimisation of the cost of communication between VMs, (iii) increase in the data centre energy-efficiency and (iv) improvement in the data centre scalability. They adopt a fuzzy mechanism to unify different optimisation objectives and compare the performance of some algorithms (i.e., random and application aware) on the resulting problem when applied on data centres with multiple network architectures (i.e., Tree, VL2, FatTree and BCube).

**Independent Objectives** and considering each objective on its own is what is called a pure multi-objective. Pires and Barán [175] estimate in their survey
that only 4% of the surveyed papers are using a pure multi-objective VM allocation approach (keeping the objectives independently without aggregating them). The few works that are not combining the objectives are most of the time either considering systems of small and unrealistic scale or limiting their study to two objectives.

The most recurrent works in the literature define the problem as a multi-objective one with objectives considered as independent but then propose a solution that groups (aggregates) them using some scalarisation or fuzzy logic. Unlike aggregation techniques, these types of works only group the objectives into a single one to quicken the conversion of the optimisation and do not use that objective to evaluate their approaches. Instead, they use the original objectives independently when reaching the evaluation stage, but the evaluation is not based on multi-objective metrics that capture different aspects of the Pareto front. The work of Xu and Fortes [184] is the most known in the literature and belongs to that group. They model the VM reassignment as a multi-objective problem and optimise simultaneously the total resource wastage, the power consumption and the thermal dissipation. They propose a Grouping Genetic algorithm (MGGA) which unlike most multi-objective genetic algorithms does not use a dominance based ranking, but rather a ranking based the fuzzy aggregation of the multiple objectives. They compare their algorithm against six bin-packing heuristics (mostly from the First Fit family) on relatively average sized datasets (PMs from 50 to 1000 and VMs from 100 to 2000), and show that their algorithm achieves good results on the independent objectives and scales well.

There are some works that propose algorithms that do not combine the objectives in any way. For instance, Zheng et al. [185] model a VM reassignment problem as a multi-objective allocation problem while keeping the objectives separately. They consider several objectives at the same time: the power consumption of their infrastructure, the resource wastage, the unevenness of resource utilisation in the servers, the network communications between the VMs, the network traffic dedicated to storage and the VMs migration time. They design a bio-geography based algorithm to optimise their problem. Their algorithm is a particular evolutionary algorithm which studies the geographical distribution of reassignment solutions to create an index that it then uses in the evolution. For the validation of their approach, they simulate a small system of 50 PM for hosting VMs and 10 others dedicated to storage, and only
2.2 Related Work

200 VMs. They also randomly generate the initial placement of VMs in PMs. Li et al. [186] also treat each resource as a separate dimension and convert the load-balancing of multiple resources to a complex system of objectives to optimise. They also propose a hybrid algorithm for the reassignment of VMs called MOVMrB that does not aggregate the objectives. Their approach applies an interval optimisation method to avoid useless VM migrations and thus reducing the number of migrations and decreasing the migration cost. Although this work keeps the objectives independent, they only use a small scale dataset of only 50 PMs and 200 VMs for the validation of their approach, which is moreover unrealistic given that the initial assignment is randomly generated.

In addition to evaluating the performance of their algorithms on a small and unrealistic dataset, some of the works (e.g., the works from Zheng et al. [187] and Li et al. [186]) are lacking some in the evaluation phase as they do not use multi-objective metrics. They instead judge the performance of the algorithms on objectives taken individually and not as a Pareto front.

There are few works that have a full multi-objective process from the modelling of the problem, to the design of the optimisation technique, to the evaluation phase. However, they often consider only two objectives and do not take the effects of migration into account. Pires and Barán [176] claim to be the first to provide a formulation to the VM placement as a pure multi-objective problem without aggregating the objectives in a previous work on which the current one is based on. They consider in their work both the energy consumption and the network traffic between the VMs. However, they do not consider the implications of the VM migration and their model can be considered as an assignment one. They also take into account the impact of the VM placement on the global revenue of the data centre. The authors propose a multi-objective memetic algorithm to optimise their problem and they validate it against an exhaustive search of all the feasible solutions. They use in their validation very small datasets that they generate randomly and that contain at most 10 PMs and 20 VMs. Unfortunately, the authors do not compare their approach against any other technique and do not apply multi-objective metrics to judge the quality of their algorithm.

Adamuthe et al. [188] address the multi-objective initial VM placement and try to optimise independently the overall profit, the balancing of the load and
the resource wastage of their centralised DC. They compare different genetic algorithms that do not require aggregating the objectives in any way for the optimisation of their problem, including the most known and widely compared technique in the community i.e., NSGA-II [74]. However, the authors do not consider the common constraints DCs are subject to and evaluate the performance of their algorithms on a rather tiny dataset made of 6 PMs and up to 20 VMs. The small size of their datasets made the enumeration of the non-dominated solutions in their paper (on a table) possible. Given the enumeration of the Pareto front solutions, the authors only use a visual comparison rather than multi-objective performance indicators/metrics to compare their results – which is not realistic in larger scenarios.

Gao et al. [189] tackle the VM reassignment and consider the optimisation of only two different objectives (i.e., the sum of resource wastage and the power consumption) and their problem lacks even common constraints between the VMs. They propose an Ant Colony System algorithm (VMPACS) that is built on another ACS that was designed to potentially handle data centres of large scales. However, their simulation is based on randomly generated datasets of rather small sizes that are composed of up to 200 VMs and the same number PMs so that the worst placement of VMs in PMs (the assigns one VM per PM) is also feasible. Unlike in [176, 188], the authors use two multi-objective metrics in their evaluation: (i) the first one that they call the Overall Non-dominated Vector Generation (ONVG) which counts the number of non-dominated solutions, and (ii) the Spacing (SP) which measures the average inter-distances between the non-dominated solutions. They compare their algorithm against several others ones including the aforementioned MGGA from Xu and Fortes [184] and show a significant improvement over them.

The work by Zhu et al. [190] is one of the most recent in this category. They model the initial assignment of VMs to PMs as a pure multi-objective one with the aim of reducing the total resource wastage and minimising the power consumption of their data centre. They propose an Ant Colony Optimisation algorithm which considers the load-balancing of resource consumption as a way of managing the pheromones (thus not aggregating the objectives). Similarly to Gao et al. [189], they evaluate the performance of their approach against MGGA from Xu and Fortes [184] and use the same multi-objective metrics (i.e., ONVG and Spacing). They use the dataset proposed by Xu and
2.2 Related Work

Fortes [184] in their evaluation, but do not compare against the algorithm proposed by [189] (i.e., VMPACS).

Although the aforementioned works consider the migration as part of the optimisation process, they either consider an off-line VM migration (no effect for VM migrations) or over-provision resources and ignore the effects of live VM migrations on transient resources (see Section 2.1.2.3) as it is the case in a properly defined VM reassignment problem. For example, in the work of Gao et al. [189], the authors deal with the effects of migration by restricting the load of the physical machine to exceed a predefined ratio of the capacity, even if the PM is not concerned with migrations.

The multi-objective VM reassignment is a novel optimisation problem. It is largely inspired by the machine reassignment problem proposed by Google, but considers objectives that are relevant to data centres’ managers in a non-aggregated fashion. I proposed in works that are part of this thesis a three-step method [6] to address the problem, using a modified version of the Google ROADEF/EURO challenge 2012. In [4] and [3] I proposed a linear formulation of the problem and studied the relevance of using MILP and Constraint-Based LNS (CBLNS) solvers. I showed that MILP solvers (e.g., IBM ILOG CPLEX) can only solve small instances, whereas CBLNS achieves worse results than MILP solvers on small instances, but scales well to larger ones.

There is also a category which started to rise recently and considers the optimisation of four or more objectives independently in what is called many-objective VM (re-)assignment. This category is mostly the same as the pure multi-objective optimisation described earlier. However, authors in many-objective optimisation argue that techniques that are proven efficient on multi-objective problems are based on ranking techniques and do not perform well in their context.

López-Pires [191] claim that they provide the first formulation of the many-objective VM allocation problem. They consider the initial assignment of VMs into PMs or their reassignment, with the aim of optimising five objectives independently: (i) electricity consumption, (ii) network traffic between VMs and (iii) revenue of the data centre, (iv) QoS and (v) balancing network load. However, the last two objectives are substituted with the number of VM migrations and the network traffic overhead in the reassignment case. Although they use a large number of objectives (including migration overhead), they do not take
into account any common constraint, not even the transient resources during VM migration. The authors propose a memetic algorithm that optimises the problem without aggregating the objectives. However, their work does not perform any comparison against other algorithms. They only refer to one of their other works [192] that studies the behaviour of their algorithm in the reassignment context and which only shows the correctness and scalability of their approach, with only 10 PMs and 100 VMs. In another work from the same research team [193] as the previous authors, they perform a thorough evaluation of their approach against rather simplistic First Fit heuristics and on a small data centre of 10 PM and 100 VMs. They show that their memetic algorithm outperforms the rest of the algorithms. However, despite using the term reconfiguration, they do not consider the effect of live migration either as constraints nor as objectives in their problem. Instead, they just have a short discussion on the number of VM migrations and the amount of RAM migrated in the reassignment process. Furthermore, they compare the performance of their algorithms based on the scalarisation of the objectives through a weighted sum.

Although a large number of objectives means providing decision makers with more perspectives to evaluate the solutions upon. Purshouse and Fleming [77] show in their work that a large number of objectives decreases drastically the performance of evolutionary algorithms. It is even worse knowing the scale of the data centres I am addressing in my work. Furthermore, they claim that decision makers tend to favour a small number of dimensions.

2.2.3.3 Distributed Data Centres

As it is the case in Section 2.2.2, I also categorise the works in multi-objective VM consolidation into three categories according to the decentralised data centre delivery model: (i) federations of clouds, (ii) multi-clouds and (iii) decentralised data centres. I also include the hybrid model that is half way between private and public data centres.

**Federated Clouds** is the multiple cloud delivery model where I notice the least amount of multi-objective VM consolidation work according to my literature review. This is mostly due to the creation of Cloud federations based on
predefined contracts/agreements with a static objective directly implemented into the federation management middleware. Given that the Clouds taking part in the federation are often competing and have different perspectives, the objective in this middleware is rarely modified. This makes works with a pure multi-objective formulation/optimisation rare and hard to apply unless the Clouds are highly cooperative as it is the case in academic and scientific federations of HPC.

Although federations middlewares handle a single objective in the VM assignment and reassignment, this objective is often the result of the aggregation of several sub-objectives. For example, Coutinho et al. [194] optimise the allocation of VMs in federated scientific data centres according to a weighted sum of two sub-objectives: the finish time of the planned jobs and the cost of purchasing VMs from the different data centres. They propose GraspCC-fed (i.e., an adaptation of the GRASP greedy algorithm to the federation of clouds context) for the optimisation of their problem. They also compare it against the state-of-the-art algorithm SciDim and show interesting improvements both in terms of financial cost and finish time.

To the best of my knowledge, Phan et al. [195] are first to study a pure multi-objective VM reassignment in federations of clouds with their proposed Green Monster framework. Their framework reassigns dynamically a set of services (or a set of VMs) between data centres located in nine Western European countries, with the aim of increasing the consumption of renewable energy, reducing energy consumed for cooling and decreasing the latency between VMs and their users. They use NSGA-II in their framework and show in a simulation a good improvement with regards to the individual objectives and the hypervolume. However, they do not compare their technique against other multi-objective algorithms.

The Entice project [196] (a Horizon 2020 European project) also makes the exception as they design a framework for the VM management in federations of clouds which takes into account two objectives separately and handles: (i) the initial VM placement, (ii) the off-line reassignment and (iii) the on-line VM migration. However, their approach is designed with only these two objectives: the cost of the VM and its performance. Kimovski et al. [197] leverage the Entice framework. Similarly to [195], they propose NSGA-II as the optimisation engine to optimise the VM assignment with the same two
aforementioned objectives. The authors show in their study the behaviour of the used framework when dealing with different federation sizes, but do not compare different multi-objective optimisation techniques.

Most of the literature in the multi-objective optimisation in federations of clouds do not consider the (re-) assignment of VMs, but instead the scheduling of tasks with short durations that are often dependent on each other. For instance, Iturriaga et al. [198] propose a multi-objective algorithm for the scheduling of tasks and then evaluate it based on multi-objective metrics. Their problem looks similar to the broker’s one, but given that they have the control over the infrastructure of the Clouds, it is considered as federations of clouds. The authors aim in their work to optimise SLA violations, energy consumption and makespan of jobs (time between the arrival of a job and finishing its execution). They propose to embed the Clouds that are part of the federation with distributed schedulers that are exclusively dedicated to optimising the makespan, while keeping a general manager at the federation middleware. They compare in their work two multi-objective evolutionary algorithms i.e., NSGA-II and MOCell (a multi-objective GA) against several other heuristics such as the Round Robin, Min-Max and Max-Min on a combination of multi-objective quality indicators including the hypervolume. They show in their experiment that the evolutionary algorithms have the best overall performance while not outperforming each other. Another work from the same team [199] address a similar task scheduling problem in federated data centres as in [198]. However, they target federations of clouds that are potentially powered by renewable energy sources with a limited availability. They also take into account different objectives: QoS, resource usage and electricity consumption of both PMs and cooling systems. However, the paper only shows preliminary results with NSGA-II and does not put forward any results with multi-objective metrics.

**Multi-clouds** also attract studies dealing with multiple objectives. However, given that they do not require a prior agreement and that VM commissioning in this context goes through the default APIs of public clouds, there are not as many objectives a multi-cloud broker might be interested in optimising. The majority of the works mostly consider the cost of the VM allocation, the respect of SLAs, the latency to the users or the makespan/completion time as objectives to optimise.
2.2 Related Work

Jiao et al. [200] try to reconfigure the placement of services and users’ data of multi-layer web services where users can publish and access data of other users. The authors reassign their data and services between the different public clouds while optimising the aggregation of multiple objectives using a weighted sum. They also consider the optimisation of four objectives. The three first objectives are usual in this context: the traffic between the VMs, the distance between the layers of the same service and the cost of migration. The last one is unusual as it aims at reducing the Carbon footprint of their services (not the entire Clouds), which the authors claim is part of what they call ‘a socially aware service’. They put forward an algorithm that iterates through two routines: a greedy algorithm and a set of graph cuts. They run an experiment with a service that has over 100,000 users of a real geographic distribution, placed in 10 different Clouds across the USA, and show that their approach improves significantly the initial placement on all objectives. They also show that their approach keeps improving the initial state even when given an objective with different weights.

Frey et al. [201] study the migration problem of services sitting on top of VMs from a private data centre to different public clouds. They optimise the reassignment of VMs to public clouds based on three objectives taken separately: (i) the cost of hosting the VMs, (ii) the response time, and (iii) the number of SLA violations. However, the authors consider an off-line migration and do not model the migration effects. They propose a genetic algorithm called CDOXplorer which leverages NSGA-II for the multi-objective VM reassignment that they evaluate against two sampling techniques and a regular genetic algorithm on a multi-cloud of three public clouds. Their evaluation using multi-objective quality indicators shows that they outperform the other algorithms by up to 60%. El Kateb et al. [202] also studies the multi-objective VM reallocation to public clouds. Unlike in [201], they take five independent objectives (i.e., a many-objective problem) into account: (i) cost, (ii) security classification of the Clouds, (iii) deployment of all the layers of an application, (iv) CPU overloading, and (v) SLA satisfaction. They propose two different algorithms (i.e., CMOP and CMOP-epsilon) which are both based on the same genetic algorithm, but differ in the selection of individuals between different generations. CMOP-epsilon uses the dominance in objectives and the crowding distance [74] (similar to NSGA-II) while CMOP computes the average of the objectives and ranks them according to that average value. The authors
show in their comparison that CMOP is better but tends to favour two objectives over the others, whereas CMOP-epsilon achieves good results with a high diversity and trade-offs between the objectives.

Frey et al. [201] study another similar and important problem as they consider a reassignment of VMs between the public clouds based on a predictive load model. They aim in their work to independently optimise the overall cost, the simplicity in configuration (the lesser IaaS platforms to manage the better) and the migration effort (similar to the number of VM migrations). They propose an evolutionary algorithm with multiple operators (i.e., crossovers and mutations) that are specifically designed to their problem, and test it on two real-life datasets (i.e., lowWeb and highWeb), each of them containing five instances. They evaluate the performance of their algorithm against a random reassignment algorithm and a theoretical lower bound on the independent objectives, and show that their approach optimises well all the considered objectives, and even reaching the theoretical lower bounds on some cases.

Multi-clouds are sometimes leveraged by data centres and clients with punctual, short-term and dependent tasks/VMs (i.e., with constraints of precedence given as directed graphs). Therefore, brokers express their problem in this scenario as a scheduling one, where it is no longer question of running all the tasks at ones but rather their scheduling within the brokered PMs from the different public clouds. For instance, Somasundaram and Govindarajan [149] aim at optimising the scheduling of VMs with deadlines on the execution of their processes that are part of HPC applications in the Cloud. They aim at minimising the execution time of the VMs, the brokering cost and the number of rejected VMs. However, given the on-line and dynamic aspect of their brokering, they aggregate these objectives and optimise their weighted sum using a Particle Swarm Optimisation algorithm. They compare their algorithm against three other algorithms (i.e., a genetic algorithm, an ant colony optimisation algorithm and a rank-based allocation heuristic. They based their comparison on a simulated testbed of a maximum of 500 PMs and up to 1000 jobs and show that their algorithm achieves better results on all the considered objectives. The work by Panda and Jana [203] is another example of the multi-objective scheduling in multi-clouds. The authors model their problem with three independent objectives (i.e., makespan, total cost and the average time the Clouds are busy being utilised) and do not aggregate them. They propose a scheduling algorithm that combines two phases (i.e., a normalisation
of objectives and a Min-Min algorithm) which they call MOTS, and compare it against two single objective algorithms: (i) Cloud Min-Min Scheduling which only aims at minimising the makespan and (ii) Profit Based Task Scheduling which only aims at increasing the profit. As it was expected, they show on two benchmarks (512 and 1024 jobs, 16 and 32 Clouds each) and one synthetic dataset (from 100 to 1000 jobs, from 4 to 40 Clouds) that their algorithm achieves better trade-offs than the two other approaches. However, the authors do not base their comparison on multi-objective metrics. Instead, they use the values on the individual objectives.

**Decentralised Data Centres** appear mainly in large Enterprises. Managers of these large Enterprises often have different objectives on how to make their multiple hosting departments better \[12\] (e.g., more energy efficient, more reliable, etc.). At the same time, given the relative autonomy of their departments and their freedom for managing their own infrastructure according to their preferences, managers have to encompass this when planning their VM reassignment.

As it is the case with the single objective VM reassignment in decentralised data centres, grid computing also addressed a similar problem: the multi-objective scheduling of tasks in two-level hierarchical grid systems. The work of Kurowski et al. \[204\] is one example. The authors consider a system with two scheduling levels: (i) the grid broker level which allocates the jobs to the PMs, (ii) and the local schedulers at the PMs which schedule the processing of these jobs according to their predefined configuration. The authors take preferences of several stakeholders into account at the broker level such as the completion time and makespan, but aggregate them using the Ordered Weighted Averaging operator to only optimise a single objective problem at the end. They evaluate the performance of three different heuristics for the optimisation of their top-level scheduling (i.e., Min-Load, Min-Parallel-Load and an augmented Load-Balance) depending on the number and duration of the jobs and show that their augmented Load-Balance achieves better results on most of the objectives, but its results are of a high standard deviation. The work by Gkoutioudi and Karatza \[205\] is another example of a multi-objective job scheduling in two-level hierarchical grid systems. The authors address a similar problem as Kurowski et al. \[204\]. However, they aim at independently
optimising the makespan, the probability of failure and the energy consumption. The authors propose a modified genetic algorithm for the scheduling at the grid broker level. Their GA uses a greedy initial population, applies an aggregated fitness function for the selection of good solutions, and takes its execution overhead into account. Despite keeping the objectives independent, the authors compare their GA against three other algorithms (Min-Min, Max-Min and Shortest Queue) on each objective separately and do not use multi-objective quality indicators.

In the context of data centre optimisation, some works consider geographically distant data centres in a centralised fashion, without taking into account the autonomous aspect of different data centres. This is the case in my works [3, 6] (part of this thesis), where different locations are gathered under the same centralised infrastructure and reassign VMs with the aim of optimising independently the reliability, energy consumption and migration cost in a totally pure multi-objective way (from modelling the problem to evaluating the performance of the different algorithms). There is also another work from Kessaci et al. [206] that considers the same type of data centres, but addresses the problem of VM scheduling. They work on the scheduling of time-limited jobs that are part of HPC applications in the PMs located in geographically distant data centres. They propose a genetic algorithm called MO-GA that optimises independently the energy consumption, CO$_2$ emissions and the global profit. They evaluate the evolution in performance of MO-GA when varying its different parameters and the type of datasets, however, the evaluation is done on the objectives taken separately and not using multi-objective quality indicators.

To the best of my knowledge, my work is the first to introduce the multi-objective VM reassignment problem in decentralised data centres in [152] and to model the problem and design a solution to address it in [5] (these works are part of this thesis). In [5], I designed a two-level system: a reassignment level to decide in which department to reallocate every VM, and a placement level (one placement module per autonomous department) to place the VMs in their assigned departments according to the department local preferences. Each placement module uses a combination of First Fit and Hill Climbing algorithms to optimise the VM placement according to the locally preferred combination of objectives, while the reassignment module considers three objectives independently (i.e., reliability, electricity cost and migration cost) and
leverages a multi-objective hybrid metaheuristic (i.e., GeNePi) that is designed in this thesis to optimise the reassignment. In this work, I also consider the overheads led by VM migrations on transient resources. I compare my system to several other ones based on multi-objective metrics and show that my system outperforms them both quantitatively and qualitatively.

**Hybrid Data Centres** is the context where the research community is very active in the recent years when it comes to multi-objective VM allocation. However, most of the works are formalising the problem as a scheduling problem with VMs/jobs running for a limited duration.

Raju et al. [207] propose an algorithm that is inspired by the behaviour of bats (i.e., echo-localisation and hibernation) to schedule VMs while minimising the energy consumption of the private data centre and reducing the execution time. The authors keep their objectives independent without aggregating them. However, they do not perform any evaluation of the performance of their approach.

Hu et al. [208] use a genetic algorithm called NSjDE for the job scheduling problem in hybrid data centres that tries to limit its number of evaluations in order to fasten its optimisation. NSjDE optimises both the tasks completion and the cost. They compare their approach against three other algorithms (i.e., a Min-Min and two other genetic algorithms). They show in their experiment with 10 hosts and up to 300 jobs that their algorithm is not only faster but also achieves better results on both considered objectives, without using multi-objective metrics though.

Zuo et al. [209] extend the initial definition of the scheduling problem and add deadlines to jobs completion and aim at optimising completion time, QoS and profit. They also propose MOSACO; a multi-objective ant colony algorithm with an entropy measure to address their problem, that they then compare against four other heuristics (i.e., Cost-First, Time-First, Min-Min and First-In First-Out) using Cloudsim on a data centre with a centralised private cloud and two public clouds. Their experiment shows that MOSACO achieves the best results on the individual objectives in terms of completion time, while Cost-First has a clear advantage in terms of cost.
2.3 Conclusion

I presented in this chapter the background that is useful for the understanding of this thesis. I also presented and described the work related to mine, both in data centres with different delivery models, and in single objective and multi-objective scenarios.

In the background section, I started by presenting the virtualisation concept with a brief comparison of two main techniques: VMs and containers and I explained the advantage of leveraging VMs for the consolidation of data centres. Then, I presented different scenarios and contexts for the VM consolidation going from the assignment to the reassignment in its two different types (i.e., reactive and planning evolutions). Next, I described the different services that are provided by public clouds and more particularly IaaS. I finished by explaining the difference between multi-objective resolution (exact) and multi-objective optimisation (not necessarily exact) and the different metrics available to evaluate algorithms’ performance in the latter.

I surveyed more than 130 papers in the related work. The first part surveys works related to the single objective VM reassignment problem in centralised data centres, with the different works: before, during and after the Google ROADEF/EURO 2012 challenge. Whereas, the second part is dedicated to the works related to the VM reassignment problem in different distributed data centre delivery models (i.e., federations of clouds, multi-clouds, decentralised data centres and hybrid clouds) and the difference between them. The last part focuses on the multi-objective optimisation of the VM reassignment. I started by presenting the most commonly used objectives. Then, I presented works related to the multi-objective VM reassignment in centralised data centres with their differences. After that, I presented the works on the same problem, but in the different delivery models of distributed data centres.

In the next chapter, I model the multi-objective VM reassignment in centralised data centres as a linear problem. I also study the applicability of a linear solver and one of the best performing algorithms in the Google ROADEF/EURO challenge 2012 and evaluate their performance.
EXACT AND HYBRID ALGORITHMS FOR THE
MULTI-OBJECTIVE VM REASSIGNMENT PROBLEM

In this chapter, I provide a full linear formalisation of the multi-objective VM reassignment. I also study the applicability and evaluate the performance of both a linear solver (i.e., IBM ILOG CPLEX) and one of the best performing algorithms on the Google ROADEF/EURO Challenge 2012 (i.e., CBLNS).

3.1 Introduction

Data centres are facilities dedicated to hosting many computer resources. They evolve constantly as for instance PMs age and are eventually decommissioned, new ones are bought regularly, and hosted VMs are updated to potentially greedier ones. Decision makers and managers of data centres adapt their systems to these evolutions and migrate VMs from one PM to another following technical and non-technical constraints and preferences. This is called reassignment of VMs to PMs. For instance, managers may want to increase the reliability of their data centres and move the workload from overloaded PMs to less loaded and/or more powerful ones. Often, they also try to move the
workload to power efficient PMs, in order to lower the cost and environmental impact of the data centres.

One problem is that PMs can range to up to tens of thousands, and services up to millions. At this scale, any instance of the reassignment problem becomes a challenge to the existing heuristics and solvers, and finding the ‘best’ (re-)assignment an illusion. Another problem is that, as 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. [12] describe such an enterprise environment where managers of hosting departments have various perspectives when it comes to placement decisions.

CP and MILP are known to be inefficient for large scale problems with a limited execution time [111], and usually researchers focus on other optimisation techniques (e.g., local search [105, 107] or greedy algorithms [115]) or mix CP or MILP with some other optimisation solutions (e.g., local search [111]) – but often with little or no success.

In the first part, I consider that data centres are centralised (see Figure 3.1) and decision makers have full control over which VM is placed on which PM. Hence I call the problem I address in this chapter Multi-objective VM Reassignment Problem in centralised data centres. Notice that despite considering the data centre as one block, PMs can still be in geographically distant locations.

In this context of centralised data centres, CP, MILP or Large Neighborhood Search (LNS) do not seem promising approaches, as the search space is large and the constraints hard and complex. As far as I know, the only related work tackles the problem using some other optimisation techniques. Mehta et al. [111] use CP on a relaxed problem, not the original VM reassignment one, which leads me to think that CP is inefficient for my problem – and anyway I also tried IBM ILOG CP Optimizer (a commercial CP solvers) and noticed extremely poor performances (i.e., the solver could only find feasible solutions on the smallest data centre I use). One of the challenges here is that
3.1 Introduction

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. Therefore, as it is commonly accepted by practitioners and in the literature [19, 111], I use a time limit for the multi-objective VM reassignment problem in centralised data centres.

However, given that CPLEX’s solutions are generally of a better quality than the ones of other optimisation techniques, and that there are several relaxation mechanisms in CPLEX and the multi-objective problem itself, I think that it is important to study whether an MILP solver, such as CPLEX, can be used for the multi-objective VM reassignment problem in centralised data centres. 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 I believe are worth exploring.

In this chapter, I first start by providing a full linear definition of the multi-objective VM reassignment problem in large centralised data centres (Section 3.2). After that, I describe the experimental setup (Section 3.3). Then, I perform a thorough study of how suited is an MILP solver (CPLEX) and show that it is useful only for rather small or medium scale data centres and

Figure 3.1: Overview of centralised data centres.
with some relaxations: a certain tolerance gap and a limited number of directions explored in the search space (Section 3.4). Next, I give another exhaustive study of whether CBLNS (a combination of LNS and a CP solver) can optimise the multi-objective VM reassignment problem in centralised data centres and what parameters give the best solutions and show that there is an optimal number of directions that we can explore in the search space (Section 3.5). Finally, I make some concluding remarks (Section 3.6).

3.2 Problem Definition

The Multi-objective VM Reassignment Problem in large centralised data centres consists of optimising the usage of a set of PMs $\mathcal{M}$ according to various objectives. Any reassignment has to satisfy constraints (often in large number) of the system and find a new PM $M(v)$ for every VM $v$ in the system, initially placed in PM $M_0(v)$. The multi-objective VM reassignment tries to find non-dominated solutions (better than every other solution in some directions of the space). In some cases, $M_0(v) = M(v)$, which means that the VM $v$ does not move during the reassignment. In this section, I propose the first linear formulation for the multi-objective VM reassignment by extending the single objective model that was originally proposed by Mehta et al. [111]. I first describe the different elements of data centres, followed by the linearised constraints of the problem, and I finish with the different linearised objectives that I believe are the most relevant (note that this approach is agnostic to objectives and would work with any other linear objective function).

3.2.1 Problem Description and Notation

A centralised data centre is composed of a set $\mathcal{M}$ of PMs. Each PM $m_i \in \mathcal{M}$ has a finite amount $Q_{m_i,r}$ of resource $r \in \mathcal{R}$ (e.g., CPU, RAM, storage). PMs $m_i \in \mathcal{M}$ belonging to the 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 ($r \in \mathcal{T} \mathcal{R} \subseteq \mathcal{R}$, 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 ($r \in \overline{\mathcal{T} \mathcal{R}}$, e.g.,
3.2 Problem Definition

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 the same multi-tier application are usually gathered by services \( \mathcal{S} = \{s_1, \ldots, s_p\} \), with \( s_p = \{v_{p,1}^1, \ldots, v_{p}^p\} \).

3.2.2 Constraints

I present here the linear constraints of my problem.

3.2.2.1 Reassignment Constraints

Consider a binary variable \( x_{v,m} \) for every VM \( v \in \mathcal{V} \) and for each PM \( m \in \mathcal{M} \), which is set to 1 if \( M(v) = m \) and 0 otherwise. Constraints (3.1) ensure that every VM is reassigned to one and only one PM:

\[
\forall v \in \mathcal{V}, \quad \sum_{m \in \mathcal{M}} x_{v,m} = 1 \quad (3.1)
\]

3.2.2.2 Capacity Constraints

Capacity constraints describe the resource capacities of the PM \( m \) as limiting the resource demands of the VMs \( v \) hosted on them. There are two ways of computing resource utilisation \( U_{m,r} \) of a PM \( m \in \mathcal{M} \) and a resource \( r \in \mathcal{R} \): (3.2) for non-transient resources and (3.3) for transient resources.

\[
\forall r \in \mathcal{TR}, \forall m \in \mathcal{M}, \quad U_{m,r} = \sum_{v \in \mathcal{V}} d_{v,r} x_{v,m} \quad (3.2)
\]

\[
\forall r \in \mathcal{TR}, \forall m \in \mathcal{M}, U_{m,r} = \sum_{v \in \mathcal{V} | M_0(v) = m} d_{v,r} x_{v,m} + \sum_{v \in \mathcal{V} | M_0(v) \neq m} d_{v,r} x_{v,m} \quad (3.3)
\]
The total resource utilisation of a PM $m \in \mathcal{M}$ should not exceed its capacity $Q_{m,r}$ for every $r \in \mathcal{R}$:

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

(3.4)

### 3.2.2.3 Conflict Constraints

Services/applications are often multi-tier (e.g., to separate concerns) and replicated (for performance and security reasons), so it is realistic to assume here that VMs (the atomic element of workload) are organised by services. It is common for services to be in conflict and have an anti-cohabitation constraint [210]), i.e., the VMs composing a service cannot share the same host – for some reliability, security and performance reasons. Therefore, VMs which belong to the same service have to be reassigned to different PMs.

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

(3.5)

### 3.2.2.4 Dependency Constraints

Services can also depend on each other and in this case the VMs of these services need to be close to each other – to increase the performance of the system. Of course, as the dependencies between services can be complex, the assignment can be tricky: a VM $v \in \mathcal{V}$, 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 PM in $N(m)$ with $\exists v' \in s_j \mid M(v') \in N(m) \cap \exists v'' \in s_k \mid M(v'') \in N(m)$.

Let $\mathcal{D}$ be the set of service dependencies such that $\mathcal{D} = \{(s_i, s_j) \mid s_i, s_j \in \mathcal{S}$ and $s_i$ depends on $s_j\}$, then:

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

(3.6)

To give a linear definition of constraints (3.6), I introduce the binary variables $y_{s,n}$ for every service $s \in \mathcal{S}$ and for each neighbourhood $n \in \mathcal{N}$. Constraints (3.7) and (3.8) ensure that each variable $y_{s,n}$ is set to 1 if at least one
3.2 Problem Definition

VM from the service $s$ is hosted by a PM in the neighbourhood $n \in \mathcal{N}$ and to 0 otherwise:

$$\forall s \in \mathcal{S}, \forall n \in \mathcal{N} \sum_{v \in s} \sum_{m \in n} x_{v,m} \leq |\mathcal{N}|.|\mathcal{S}|.y_{s,n} \quad (3.7)$$

$$\forall s \in \mathcal{S}, \forall n \in \mathcal{N} \sum_{v \in s} \sum_{m \in n} x_{v,m} \geq y_{s,n} \quad (3.8)$$

If a service $s_i$ depends on $s_j$, constraints (3.9) guarantee that there is not any VM from $s_i$ reassigned to a PM in a neighbourhood $n \in \mathcal{N}$ without having at least one VM from $s_i$ reassigned to that neighbourhood:

$$\forall (s_i, s_j) \in \mathcal{D}, \forall n \in \mathcal{N} y_{s_i,n} \leq y_{s_j,n} \quad (3.9)$$

### 3.2.2.5 Spread Constraints

For the same reasons of reliability, security and performance, services require that the number of locations hosting at least one VM has to be greater than a certain number, called spread number. This allows increasing the resilience in case of failure of a data centre: the bigger the spread number, the safer the service. Let me introduce a binary variable $z_{s,l}$ for every service $s \in \mathcal{S}$ and for each location $l \in \mathcal{L}$ that composes the centralised data centre. Constraints (3.10) and (3.11) ensure that $z_{s,l}$ gets the value 1 if the service $s$ has at least one VM running in a PM at the location $l$:

$$\forall s \in \mathcal{S}, \forall l \in \mathcal{L} \sum_{v \in s} \sum_{m \in l} x_{v,m} \leq |\mathcal{N}|.|\mathcal{S}|.z_{s,l} \quad (3.10)$$

$$\forall s \in \mathcal{S}, \forall l \in \mathcal{L} \sum_{v \in s} \sum_{m \in l} x_{v,m} \geq z_{s,l} \quad (3.11)$$

Constraints (3.12) force every service $s$ to run on at least $spreadNumber_s$ number of locations:

$$\forall s \in \mathcal{S}, \sum_{l \in \mathcal{L}} z_{s,l} \geq spreadNumber_s \quad (3.12)$$
3.2 Problem Definition

Figure 3.2 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 (VM Reassignment) An assignment $A$ of VMs to PMs is a mapping: $A : V \mapsto M$, such that $A(v, M) \to m$, which satisfies the constraints (3.1–3.5 and 3.7–3.12).

A reassignment is a function: $ReA : A \mapsto A$ which returns a new assignment for a given initial assignment of VMs to PMs.
3.2 Problem Definition

3.2.3 Objectives

As it is aforementioned, there are several perspectives on the best optimisation, which translate in my case into several objectives. Some studies [77] show that a large number of objectives decreases drastically the performance of evolutionary algorithms, and decision makers tend to favour a small number of dimensions. I focus here on three objectives: electricity cost, VM migration cost and reliability cost, as they are recognised in the literature [211–213] and make sense in practice. The multi-objective variant of the VM reassignment problem in large centralised data centres consists of minimising separately the cost functions defined by the different objectives.

3.2.3.1 Reliability Cost

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 PM, the vendors of its parts (e.g., processor maker) and the past history of similar PMs. They are complex to collect and understand, and it is not exactly known 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 [213]). One thing that is known is that as opposed to the risk of failure, the reliability is easier to compute and gathers fewer questions. PMs do operate better when they are not too loaded, and reliability can be estimated through the load: the more loaded a PM, the greater the risk of performance issues or failures.

For each PM \( m \in M \) and each resource \( r \in R \), I define the safety capacity \( \rho_{m,r} \) 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 [8]. 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 \( R_{\text{cost}} \) represents the ‘non reliability’ over the full data centre.

\[
\forall m \in M, \forall r \in R, \quad R_{m,r} = U_{m,r} - \rho_{m,r} \geq 0 \tag{3.13}
\]

\[
R_{\text{cost}} = \sum_{r \in R} \sum_{m \in M} R_{m,r} \tag{3.14}
\]
3.2 Problem Definition

Note that this definition is inspired by the concept of safety capacity introduced in the Google ROADEF/EURO challenge [19]: if one or several resources of a PM are overloaded then the PM may not be able to satisfy its SLAs. This is similar to the resource buffer that placement algorithms often assume [8].

3.2.3.2 Electricity Cost

Electricity cost of running PMs accounts for up to 50% of data centres operation costs [211] 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 of Ireland) by 2020 [214]. There is a global trend towards a greener and power-aware practices, and this will certainly lead to an increase in the electricity price and give another incentive for data centre managers to minimise their electricity consumption. Modelling electricity cost of a PM is complex but I follow the general assumption that states that it is a linear function of its CPU usage [171, 184]. I then just define two parameters, $a_m$ (fixed cost of running $m$ with $n$ load on the CPU) for every PM $m$ and $\beta_m$ (linear factor). This does not take into account other elements that may be relevant but are somehow out of the scope of my study here (e.g., cooling of data centres).

For each PM $m \in \mathcal{M}$, I introduce a binary variable $o_m$ to be set by constraints (3.15) to 1 if the PM $m$ is switched on (i.e., hosts at least one VM) and 0 otherwise.

$$\forall m \in \mathcal{M}, \quad o_m \leq \sum_{v \in \mathcal{V}} x_{v,m} \leq |\mathcal{V}| o_m \quad (3.15)$$

The electricity cost $E_{cost}$ is composed of the electricity consumption of each PM $m \in \mathcal{M}$ multiplied by its price $\gamma_m$. The electricity consumption of a PM $m$ is often modelled as a linear function of its CPU utilisation [171, 184], with $a_m$ being its electricity consumption at idle and $\beta_m$ its consumption per unit of CPU usage.

$$E_{cost} = \sum_{m \in \mathcal{M}} \gamma_m (a_m o_m + \beta_m U_{m,\text{CPU}}) \quad (3.16)$$
3.2.3.3 Migration Cost

Migrating a VM has a cost which is often neglected by research in the area but is well known by practitioners [212].

For each VM $v \in V, M$, I introduce a binary variable $mig_v$ to be set by constraints (3.17) to 1 if $v$ is reassigned and 0 otherwise.

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

(3.17)

The migration cost $M_{\text{cost}}$ concerns all migrating VMs. For each migrating VM $v \in V$, the migration cost is the time needed to: (i) prepare the VM $\mu_1(v)$ for migration, (ii) transfer its image from its initial placement to its new host $m \mu_2(v, M_0(v), m)$ and (iii) deploy it in the new host $\mu_2(v)$ [212]:

$$M_{\text{cost}} = \sum_{v \in V} \left( [\mu_1(v) + \mu_3(v)] \cdot mig_v + \sum_{m \in M} \mu_2(v, M_0(v), m) \cdot x_{v,m} \right)$$

(3.18)

All these costs are dependent on some VM 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 I do not evaluate in this thesis. Note that costs are based on what was given to participants to the Google ROADEF/EURO challenge [19]. I do not claim to be exhaustive or even totally accurate and I refer the readers to [42–44] for more details on migration costs. Here, I only assume that migration has a non-negligible cost in data centres and I use the model and values given by the challenge mentioned above.

3.3 Experimental Setup

In this section, I describe the instances of the problem (data centres to optimise) and the metrics used to judge the proposed systems. I adapted the dataset provided by Google to the ROADEF/EURO challenge [19] to fit my
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.

3.3.1 Dataset

Google and the ROADEF/EURO societies released a few years ago a dataset, now widely used in the Operations Research community, for the evaluation of VM reassignment solutions [19]. 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 I 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 my work and others using the Google ROADEF/EURO Challenge best-known results). My instances aim to model realistic scenarios as we observe them in large companies. For my evaluation, I take the 14 first instances, leaving out only the largest ones (see Table 3.1). I picked these instances only to limit the execution time of my experiments (I have 10 runs per algorithm and per instance). Two of the objectives I define are present in the Google ROADEF/EURO as cost functions: safety/reliability and migration, and I only add electricity. I randomly generate electricity consumption constants \((a, b)\) for every PM \(m \in \mathcal{M}\) and also the electricity cost \(\gamma\) 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). I 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.

3.3.2 Metrics

Comparing multi-objective optimisation approaches is complex as the set of solutions they give on a problem can be seen from different perspectives: coverage, closeness to the Pareto frontier, variety, and many more [82]. The problem probably comes from the fact that the Pareto frontier is unknown most of
3.3 Experimental Setup

Table 3.1: Characteristics of the different instances used in my evaluation.

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

the time, and that the different objectives cannot be taken in isolation to give the quality of any solution. In this chapter, I consider two unary operators as metrics (for a more comprehensive study of the various possible operators and their requirements, see Chapter 2): these unary operators take a set of solutions and give a single value, allowing to compare the different approaches. Furthermore, they either do not have any requirement or have a requirement that I could easily fulfil (see Chapter 2 for more details).

3.3.2.1 Number of Non-dominated Solutions

I use the number of non-dominated (efficient) solutions in my experiments as my first metric. I refer to it as the quantity of found solutions. This metric is highly important for data centre capital allocators as it gives them more...
3.3 Experimental Setup

choices. It also provides them with backup solutions if the preferred choice appears difficult or technically impossible to implement.

3.3.2.2 Hypervolume

The hypervolume [76] (a.k.a., $S$ metric) refers to the quality of a set of solutions. The hypervolume is by far the most used [83] metric 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 3.3 shows a two-dimensional VM 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.

![Figure 3.3: Metrics: number of non-dominated solutions (black dots) and hypervolume (grey area).](image)

Figure 3.3: Metrics: number of non-dominated solutions (black dots) and hypervolume (grey area).
3.4 CPLEX for the VM Reassignment Problem

The goal of this section is to study both performance and scalability of an MILP solver (i.e., CPLEX) on the Multi-objective VM Reassignment Problem. First, I explore how CPLEX performs on the (Mono-objective) VM Reassignment Problem, i.e., the original problem from the Google ROADEF/EURO Challenge. I show how difficult the problem is for CPLEX, even in this simpler version. Then I 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.

3.4.1 CPLEX for the Mono-objective VM Reassignment Problem

Portal et al. [109] show that the VM Reassignment Problem is too difficult for an MILP solver like CPLEX when applied on the Google ROADEF/EURO Challenge 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’s 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, I have here a larger time budget to solve the instances.

Table 3.2 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, I also add the time limit (last column) 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 an MILP solver is inefficient for this 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 3.2: 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). The symbol ‘x’ means that no solution was found in the time limit. The row b_* is for instances b_2, b_3 and b_4.

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

3.4.2 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, I 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, I 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
3.4 CPLEX for the VM Reassignment Problem

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 3.4 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 the 3-dimensional space (3 objectives), I 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 3.2 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 I 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.

Figure 3.4 – continued on next page
3.4 CPLEX for the VM Reassignment Problem

Figure 3.4 – Continued from previous page

![Graphs showing time versus number of vectors for different cases a_1_3, a_1_4, a_1_5, a_2_1, a_2_2, a_2_3.](image)

Figure 3.4 – Continued on next page
According to Figures 3.4, 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 the optimisation in a reasonable execution time, the larger/more complex is the instance the larger the optimality gap I consider. Unlike what we might think, CPLEX’s execution time is very heterogeneous and varies a lot from a vector to another (execution time curves are not linear). Thus, knowing CPLEX’s 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 CPLEX could only be run on one vector.
3.4 CPLEX for the VM Reassignment Problem

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 the 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 3.5 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 on the right side give the hypervolume for the same experiments with all solutions in the solution set. We notice from the graphs on 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 on the right of Figure 3.5. We notice that using a small optimality tolerance gap gives 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 my evaluations I collect all intermediate (good, i.e., non-dominated) solutions.
3.4 CPLEX for the VM Reassignment Problem

Figure 3.5 – Continued on next page
3.4 CPLEX for the VM Reassignment Problem

Figure 3.5 – Continued from previous page

Figure 3.5 – Continued on next page
3.4 CPLEX for the VM Reassignment Problem

Figure 3.5 – Continued from previous page

Figure 3.5 – Continued on next page
3.4 CPLEX for the VM Reassignment Problem

Figure 3.5: Hypervolume obtained on the different Google ROADEF/EURO 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 3.5 show the hypervolumes obtained with different numbers of vectors and different tolerance gaps, and this can be read together with Figure 3.4 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 5th and 7th (whenever CPLEX reaches them in the time constraint), especially when the optimality gap is tiny. This leads me to think that running several CPLEX optimisations with a large number of vectors (larger than 5) would not be as beneficial as one 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 3.4 and rightmost figures in Figure 3.5 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 achieving a good hypervolume while keeping the execution time reasonable for all the instances.

3.5 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 known to work well in a limited time.

The Google ROADEF/EURO challenge is not multi-objective and the many algorithms proposed to solve it cannot be applied directly to my 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 has been done in the previous section with CPLEX.

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

3.5.1 Description of CBLNS

As its name indicates it, CBLNS uses an LNS [215] metaheuristic for optimising the mono-objective VM reassignment problem.

Figure 3.6 gives a visual representation of the LNS algorithm. LNS starts by considering the initial assignment as the initial solution. It then selects a
3.5 CBLNS for the VM Reassignment Problem

subset of PMs and VMs from the current assignment to be reallocated, and it creates a reassignment sub-problem with only those selected PMs and VMs. This sub-problem is solved using some optimisation techniques. If the solution to the sub-problem decreases the overall cost of the global problem, then the current solution is updated by integrating the solution to the sub-problem. These steps (i.e., creating a sub-problem from a selection of PMs and VMs and updating the current solution) are repeated for as long as the allowed time is not exceeded.

Figure 3.6: Flowchart representation of the Large Neighbourhood Search Algorithm.

CBLNS is a modification of LNS that proposes a novel selection of PMs and VMs and resolution of the sub-problems.

3.5.1.1 Selection of PMs and VMs

The selection of PMs and VMs that will compose the sub-problem 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 PMs a certain number of VMs that are currently assigned to them. The total number of VMs that will be selected is also capped to $k_m^v \in \mathbb{N}^*$ per PM $m$. The value of $k_m$ is set to 1 at the beginning and is incremented at
PMs are sorted based on a utility function which takes the capacity into account. At every iteration, one PM is picked based on that order, whereas the rest (i.e., the other $k_m - 1$ PMs) are chosen randomly.

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

### 3.5.1.2 Solving the Sub-problem

Different techniques for modelling/optimising the sub-problem have been studied [111], though CP showed better results and is considered in CBLNS. Two main operators for solving the VM reassignment problem have been developed: (i) remove_VM_From_PM, which takes the VM from its current PM and reassigns it, and (ii) reassign_VM_To_PM, which sets the reassignment of a given VM to a given PM. Sub-problems 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 Google ROADEF/EURO challenge), a large number of sub-problems 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 sub-problems 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 sub-problem 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 sub-problem.
3.5 CBLNS for the VM Reassignment Problem

3.5.1.3 Implementation of CBLNS

I have adapted the code provided by Mehta et al. [111]. First, I have added the parameters required to define the electricity objective that they do not consider. Then, I have removed the resource balancing utility function that they used and was not relevant to my problem.

3.5.2 CBLNS for the Multi-objective VM Reassignment Problem

Since CBLNS is a mono-objective optimisation algorithm, I need to run it with a combination of the three considered objectives to make it suitable for my problem. As it was done for CPLEX in the previous section, I combine the objectives using a weighted sum. At every iteration, I define a vector of weights and optimise the problem using CBLNS. I consider the same vectors as CPLEX in Section 3.4.2. 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 I address here are different from the ones I evaluated before with CPLEX (see Section 3.4.2): 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. [111] 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 the execution time in my case 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, I only keep the best solution for each weight vector.

Figure 3.7 shows the hypervolume obtained using CBLNS when running with a different number of weight vectors on the different instances. I run CBLNS with every vector \( w \) for an execution time \( t_i^w = \frac{T_i}{\# \text{vectors}} \), with \( T_i \) being the time limit for every instance \( i \) as shown in Table 3.2.
3.5 CBLNS for the VM Reassignment Problem

Figure 3.7 – Continued on next page
3.5 CBLNS for the VM Reassignment Problem

Figure 3.7 – Continued from previous page

- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)

- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)

- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)

- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)
- Hypervolume (e20)

Figure 3.7 – Continued on next page
3.5 CBLNS for the VM Reassignment Problem

We see from Figure 3.7 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 Google ROADEF/EURO instance $i$, with an execution time of $T_i$ 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, and as it was expected, CBLNS also scales to large scale instances (b_2, b_3 and b_4). However, CBLNS does not perform as well as CPLEX. For instance, when using the best parameters for each of CBLNS (i.e., 5 vectors) and CPLEX (i.e., 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 see that CBLNS is only reaching 45.2%
3.6 Conclusion

I provided in this chapter a full linear definition of the multi-objective VM reassignment problem in centralised data centres, a large and difficult problem with a lot of complex constraints, and multiple reassignment objectives (i.e., electricity cost, migration cost and reliability cost). I also surveyed how classical exact and hybrid optimisation techniques, such as MILP and CBLNS, perform on it and find the best set of reassignment solutions in a limited time – the limit being quite large (10 hours for large instances). I showed that an 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). I also showed 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.

In the next chapter, I describe my hybrid metaheuristic (i.e., GeNePi) that I compare against several heuristics, metaheuristics and hybrid metaheuristics. I also propose a hybridisation of the solutions discussed in this chapter (i.e., CPLEX and CBLNS) with GeNePi as a mean to improve their performance while keeping the execution time low. Last, I evaluate the performance of these techniques against an exact resolution run for up to 30 days.
In this chapter, I compare different metaheuristics that approximate the Pareto front and propose a three-step hybrid metaheuristic which I call GeNePi. I also evaluate the combination of GeNePi with the solutions that were studied in the previous chapter (i.e., CPLEX and CBLNS), before comparing their results to an exact optimisation using the \(e\)-Constraints method run for up to 30 days.

4.1 Introduction

We have seen from the previous chapter that MILP solvers do not scale well when optimising the multi-objective VM reassignment problem in realistic and large scale centralised data centres. We have also seen that one of the best algorithms for the same problem with a single objective does not achieve a good performance when confronted with the multi-objective problem. Therefore, I investigate in this chapter other heuristics, metaheuristics, and hybrid
4.1 Introduction

algorithms as means for coping with these scaling and lack of performance issues.

I also propose a novel hybrid algorithm called GeNePi [6], which successively uses three steps: a first step (inspired from GRASP) to explore quickly all the 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 aforementioned algorithms 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.

I also investigate whether a hybrid method combining CPLEX or CBLNS (from the previous chapter) and a metaheuristic could lead to improved results, the intuition being that mixing the good aspects of each could help optimising this complex problem.

Now, while we know that one hybrid metaheuristic outperforms the other algorithms, it is difficult to assess the efficiency and effectiveness of these algorithms in absolute terms. I have implemented an exact resolution ($\varepsilon$-Constraints method [62]) of the problem in the same instances used for the comparison of the heuristic algorithms. I ran my 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 (+186%) and achieves nearly 60% of the hypervolume of the exact resolution. GeNePi also succeeds in keeping its execution time low, as it is tens of thousands times faster than $\varepsilon$-Constraints method. GeNePi is even faster or on the same order of magnitude as only one iteration of $\varepsilon$-Constraints method. We also observe that CPLEX+GeNePi does not optimise all the instances and does not find many non-dominates solutions (only 16.41% of the exact resolution). However, despite these few solutions, CPLEX+GeNePi covers 95% of the hypervolume obtained by the exact resolution on average when run for 30 days (not finishing its execution on some instances). We also see
that CBLNS+GeNePi achieves worse results than CPLEX+GeNePi, but optimises all the instances, which allows CBLNS+GeNePi to reach more than 136% in the hypervolume and more than 167% in non-dominated solutions of the exact optimisation on average when run for 30 days.

In this chapter, I first start by describing GeNePi, my algorithm for solving this problem (Section 4.2). After this, I perform an evaluation of the different state-of-the-art metaheuristics (Sections 4.3 and 4.4). Then, I propose an algorithm based on the combination of CPLEX or CBLNS and GeNePi to improve the performance both quantitatively and qualitatively, while keeping the execution time acceptable (Section 4.5). Next, I describe the $\epsilon$-Constraints method which provides an exact resolution of the problem and compare the best algorithms against it (Section 4.6). Finally, I make some concluding remarks (Section 4.7).

4.2 Description of the Solution: GeNePi

GeNePi applies successively three optimisation algorithms: GRASP (modified), NSGA-II and PLS. This idea of using three steps for an approximate resolution [216] is new in the domain of data centres optimisation.

4.2.1 Ge: a Variant of the Constructive Phase of GRASP

I use a variant of the constructive phase of Greedy Randomized Adaptive Search Procedure (GRASP) [115]. Solutions are generated by trying to reassign VMs 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 VMs according to their dependencies and their needs of resources, they are selected one by one. A decision of reassigning one per cent of the VMs from their initial hosts has been taken, because of the tightness of transient resource constraints that limits the number of reassignments. The choice of the reassignment of every VM 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 going beyond the objective types barrier,
its static definition induces getting solutions with the same objectives level of interest. This behaviour goes against the aim of a multi-objective optimisation. That is why I adopted a panel $\Lambda$ of triplet weights $(\lambda_i, \lambda_j, \lambda_k) \in [0,1]^3$, 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 PM to each VM, at each iteration. For each VM, a set of assignable PMs that respect the constraints is computed, and a value of interest is given to each PM by a weighted sum of its utility $U_i$ on each objective $i$: $\sum_{i=1}^{3} \lambda_i U_i$, which creates a set of PMs with a utility lower than or equal to \((minUtility + (1 - r) \times [maxUtility - minUtility]), with r \in [0,1]\). A random PM is selected from this eligible set to assign the VM to it. During the assignment, it may happen that a VM has no PM able to host it. The solution is declared infeasible, and removed from the initial solutions. Globally, at the end of this step, I expect to have a set of decent solutions spread over the search space.

4.2.2 Ne: NSGA-II

I use for this step a genetic algorithm called Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [74]. This step is useful for the improvement of the Pareto set 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 to get a smoother 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 creates randomly 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 [217]. In my case, crossovers consider the exchange of services (group of VMs) rather than blocks of VM
4.2 Description of the Solution: GeNePi

assignments – that minimise the number of bad crossovers. Of course the diversity is less than with crossovers on VMs, but this is compensated 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 \[74\]) , 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.

4.2.3 Pi: a Pareto Local Search

Finally, I try to improve the Pareto set by using a Pareto Local Search [218] (PLS). It consists of applying several local search operators on the solutions belonging to the Pareto frontier. A few simple moves are chosen to analyse the neighbourhood of actual solutions: (i) swap, i.e., taking two VMs and exchanging their assignment; (ii) 1-exchange, where one VM at a time is selected and reassigned to any PM that accepts it; (iii) shift, where VMs belonging to the same service exchange their assignments (which maintains the satisfaction of 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, I 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 \[74\]) 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.

\[1\] Crowding of a solution is the measure of volume between the solution which precedes and the one that follows it in every objective.
4.3 Experimental Setups

In this section, I evaluate the performance of my solution against other state-of-the-art multi-objective VM reassignment solutions. I use the same 14 instances of the problem (centralised data centres to optimise) as in Chapter 3 from the challenge Google ROADEF/EURO 2012.

I also evaluate the algorithms based on the same two metrics (i.e., hypervolume and number of non-dominated solutions) that are discussed in Chapter 3.

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.

4.3.1 Algorithms

In my study, I compare three 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 [31] and they are considered efficient. Each of them uses an ordered sequence (by resource demands) of VMs they aim to place on PMs as input. I chose among them First Fit (FF) which selects the first PM that fits for every VM; Random Fit (RF) which selects randomly a PM among those which fit; and First Fit Descent Bin-Balancing (BB) which selects the least loaded PM for each VM.

The second set of algorithms is the state-of-the-art solutions from the multi-objective optimisation field. Given the large number of existing metaheuristics, I select one algorithm for each of the common types of metaheuristics (i.e., greedy, genetic algorithm and local search). The first of them is GRASP in its original definition, i.e., the choice of reassigning VMs to PMs is based on a uniform probability distribution of the possible PMs. I also evaluate the first step of GeNePi (Ge) as it is a variation on GRASP that I expect to be better than GRASP for my scenario. The last algorithm in this family is a Pareto Local Search (PLS), with a number of boxes at every iteration equals to the number of solutions in the non-dominated solutions set. Notice that I
do not compare to a genetic algorithm (alone) here as I 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.

I also evaluate different hybrid metaheuristics: (i) GrNe where I 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 my adapted greedy algorithms (i.e., Ge), and (iii) GeNePi with its three successive steps.

4.3.2 Tuning the Steps of GeNePi

Each of the three steps composing GeNePi has several parameters that need to be tuned, and globally I need to decide how many iterations or how much time I allocate to each of them to make the best use of each. Note that the tuning has been done on one instance ($a_{1.5}$), as tuning is computationally expensive and I think the conclusions can be extended to the process in general.

![Figure 4.1: Hypervolume obtained with Ge using different values for the parameter $a$.](image)

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

Figure 4.1 shows the hypervolume obtained using Ge when setting the parameter $\alpha$ to different values ranging from 0.05 to 0.95. We see that low values of $\alpha$ lead to a bad hypervolume and that the hypervolume increases until $\alpha = 0.6$ before decreasing slightly. Therefore, the best value of $\alpha$ seems to be 0.6 regardless of the number of iterations.

![Figure 4.1: Hypervolume obtained using Ge](image)

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

Figure 4.2 shows the hypervolume obtained using Ne when setting $\alpha$ 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, I 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. I then decided to use $P_c = 0.6$ and $P_m = 0.2$.

Pi has only one parameter that can be tuned 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. 10 seems to be a good trade-off between probing a large search area and reducing the execution time. Table 4.1 summarises the tuning parameters for each step of GeNePi.

GeNePi aims at providing decision makers with an important set of good solutions, covering the solutions space, and in a reasonable time. These two ideas (quality and time) seem to be incompatible, but they just force to consider time in a different way. In particular, Ne/NSGA-II needs to have a set of good initial solutions, and thus, Ge/GRASP needs to be given enough time. It appeared from the empirical evaluations that I used to define $a$ (parameter of Ge, the first step of GeNePi) that a number of iterations from 100 to 500 leads to practically the same hypervolume. This is why I picked 100 as the number of iteration for Ge. The good number of iterations for Ne/NSGA-II is much trickier to find as it depends greatly on the quality of the initial population. It seems, experimentally, that 100 iterations with a population size of 50 give good results, so this is what I use for Ne/NSGA-II. Pi/PLS is the most time-consuming part, I use it only for one iteration in order to get a smooth Pareto.

Table 4.1: Parameters for the different steps of GeNePi after a tuning study.

<table>
<thead>
<tr>
<th>Ge ($1^{st}$ step - GRASP)</th>
<th>Ne ($2^{nd}$ step - NSGA-II)</th>
<th>Pi ($3^{rd}$ step - PLS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$ 0.6</td>
<td>Probability of crossover 0.6</td>
<td># zones (# boxes) 10</td>
</tr>
<tr>
<td>$</td>
<td>\Lambda</td>
<td>$ 4</td>
</tr>
<tr>
<td># iterations 100</td>
<td>Size of population 50</td>
<td># iterations 100</td>
</tr>
</tbody>
</table>

### 4.4 Evaluation of GeNePi Against (Meta)Heuristics

In this section I compare the average of 10 runs of GeNePi against other heuristics and metaheuristics (see Section 4.3.1) in terms of quality (hypervolume) and quantity of solutions.
4.4 Evaluation of GeNePi Against (Meta)Heuristics

Tables 4.2 and 4.3 summarise the evaluation, for all instances and all algorithms for respectively the number of non-dominated solutions and the hypervolume. 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) tend to reassign many VMs to PMs different from their initial ones. This is likely to generate violations of some of the constraints of the problem: (i) transient resources are consumed twice as much which stresses the capacity constraints, (ii) many VMs belonging to the same service have a higher chance to be reassigned to the same PM, despite the anti-cohabitation constraint, and (iii) VMs belonging to a service from which depend other services may leave the neighbourhoods they were in, leading to a violation. Therefore, FF family algorithms only work on less constrained problems and could only optimise instances $a_{1 \_1}$, $a_{1 \_5}$ and $a_{2 \_1}$.

Table 4.2: Summary of average solutions of 10 runs for the various algorithms and each of the used instances. The higher the better. I put in bold the best values for each instance.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Initial</th>
<th>RF</th>
<th>FF</th>
<th>BB</th>
<th>GRASP</th>
<th>Ge</th>
<th>PLS</th>
<th>GrNe</th>
<th>GeNe</th>
<th>GeNePi</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_{1_1}</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>87</td>
<td>20</td>
<td>42</td>
<td>10</td>
<td>14</td>
<td>106</td>
<td>224</td>
</tr>
<tr>
<td>a_{1_2}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>26</td>
<td>2</td>
<td>1</td>
<td>44</td>
<td>182</td>
</tr>
<tr>
<td>a_{1_3}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>19</td>
<td>2</td>
<td>1</td>
<td>27</td>
<td>132</td>
</tr>
<tr>
<td>a_{1_4}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40</td>
<td>2</td>
<td>1</td>
<td>75</td>
<td>136</td>
</tr>
<tr>
<td>a_{1_5}</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>32</td>
<td>16</td>
<td>112</td>
<td>282</td>
</tr>
<tr>
<td>a_{2_1}</td>
<td>1</td>
<td>33</td>
<td>41</td>
<td>1</td>
<td>69</td>
<td>57</td>
<td>4</td>
<td>71</td>
<td>152</td>
<td>231</td>
</tr>
<tr>
<td>a_{2_2}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td>2</td>
<td>1</td>
<td>41</td>
<td>197</td>
</tr>
<tr>
<td>a_{2_3}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>30</td>
<td>67</td>
<td>1</td>
<td>57</td>
<td>202</td>
</tr>
<tr>
<td>a_{2_4}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>28</td>
<td>2</td>
<td>1</td>
<td>80</td>
<td>253</td>
</tr>
<tr>
<td>a_{2_5}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>28</td>
<td>2</td>
<td>1</td>
<td>76</td>
<td>220</td>
</tr>
<tr>
<td>b_{1}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>27</td>
<td>39</td>
<td>1</td>
<td>58</td>
<td>242</td>
</tr>
<tr>
<td>b_{2}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>23</td>
<td>2</td>
<td>1</td>
<td>93</td>
<td>300</td>
</tr>
<tr>
<td>b_{3}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>108</td>
<td>1</td>
<td>43</td>
<td>162</td>
</tr>
<tr>
<td>b_{4}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td>3</td>
<td>1</td>
<td>31</td>
<td>118</td>
</tr>
</tbody>
</table>
4.4 Evaluation of GeNePi Against (Meta)Heuristics

Table 4.3: Summary of average hypervolume. $10^{\text{Exp}}$ of 10 runs for the various algorithms and each of the used instances. The higher the better. I put in bold the best values for each instance.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Exp</th>
<th>Initial</th>
<th>RF</th>
<th>FF</th>
<th>BB</th>
<th>GRASP</th>
<th>Ge</th>
<th>PLS</th>
<th>GrNe</th>
<th>GeNe</th>
<th>GeNePi</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>15</td>
<td>3.34</td>
<td>2.6</td>
<td>2.95</td>
<td>3.73</td>
<td>2.73</td>
<td>3.6</td>
<td>2.4</td>
<td>2.80</td>
<td>3.91</td>
<td>3.98</td>
</tr>
<tr>
<td>a_1_2</td>
<td>16</td>
<td>7.49</td>
<td>7.49</td>
<td>7.49</td>
<td>7.49</td>
<td>7.49</td>
<td>8.47</td>
<td>7.49</td>
<td>7.49</td>
<td>8.92</td>
<td>9.22</td>
</tr>
<tr>
<td>a_1_3</td>
<td>16</td>
<td>4.17</td>
<td>4.17</td>
<td>4.17</td>
<td>4.17</td>
<td>4.17</td>
<td>4.27</td>
<td>4.17</td>
<td>4.17</td>
<td>4.3</td>
<td>4.32</td>
</tr>
<tr>
<td>a_1_4</td>
<td>e16</td>
<td>9.72</td>
<td>9.72</td>
<td>9.72</td>
<td>9.72</td>
<td>9.72</td>
<td>11.11.1</td>
<td>9.72</td>
<td>9.72</td>
<td>12.1</td>
<td>12.17</td>
</tr>
<tr>
<td>a_1_5</td>
<td>e18</td>
<td>2.42</td>
<td>2.51</td>
<td>2.51</td>
<td>2.45</td>
<td>2.59</td>
<td>2.74</td>
<td>2.52</td>
<td>2.57</td>
<td>2.92</td>
<td>3.15</td>
</tr>
<tr>
<td>a_2_1</td>
<td>19</td>
<td>4.57</td>
<td>4.86</td>
<td>4.95</td>
<td>4.57</td>
<td>5.41</td>
<td>5.43</td>
<td>4.61</td>
<td>5.46</td>
<td>5.83</td>
<td>5.93</td>
</tr>
<tr>
<td>a_2_2</td>
<td>20</td>
<td>1.33</td>
<td>1.33</td>
<td>1.33</td>
<td>1.33</td>
<td>1.33</td>
<td>1.55</td>
<td>1.33</td>
<td>1.33</td>
<td>1.68</td>
<td>1.72</td>
</tr>
<tr>
<td>a_2_3</td>
<td>18</td>
<td>2.02</td>
<td>2.02</td>
<td>2.02</td>
<td>2.02</td>
<td>2.02</td>
<td>2.36</td>
<td>2.04</td>
<td>2.02</td>
<td>2.59</td>
<td>2.66</td>
</tr>
<tr>
<td>a_2_5</td>
<td>18</td>
<td>9.91</td>
<td>9.91</td>
<td>9.91</td>
<td>9.91</td>
<td>9.91</td>
<td>10.3</td>
<td>9.91</td>
<td>9.91</td>
<td>10.8</td>
<td>10.9</td>
</tr>
<tr>
<td>b_1</td>
<td>20</td>
<td>8.20</td>
<td>8.2</td>
<td>8.2</td>
<td>8.2</td>
<td>8.2</td>
<td>8.34</td>
<td>8.34</td>
<td>8.2</td>
<td>8.5</td>
<td>8.53</td>
</tr>
<tr>
<td>b_2</td>
<td>21</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.48</td>
<td>1.43</td>
<td>1.43</td>
<td>1.51</td>
<td>1.53</td>
</tr>
<tr>
<td>b_4</td>
<td>21</td>
<td>3.65</td>
<td>3.65</td>
<td>3.65</td>
<td>3.65</td>
<td>3.65</td>
<td>3.67</td>
<td>3.67</td>
<td>3.65</td>
<td>3.69</td>
<td>3.7</td>
</tr>
</tbody>
</table>

The same behaviour is observed for GRASP, which tends to reassign VMs instead of keeping them on their initial assignment. The decision of reassigning a VM to another PM is made by a basic draw among several relevant PMs based on a utility function for every VM. Hence the probability of keeping VMs on their initial PMs is low. Therefore, GRASP tends to generate a lot of solutions, but most of them end up being infeasible and violating one or several constraints.

The 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 – hypervolume values for PLS are always among the lowest. This comes from the fact that PLS searches for possible solutions locally, and may find some, but they are similar to the original ones and do not increase the diversity of the solutions set. For a multi-objective problem like mine, PLS is then not suited either.

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
4.4 Evaluation of GeNePi Against (Meta)Heuristics

GRASP which itself suffers from a lack of solutions.

GrNe, being dependent on the quality of the initial population, performs badly – although I give it a partial result of GRASP to help it at the start. This is a major (but well known) drawback for this algorithm, especially for my scenario for which NSGA-II is clearly not fitted: in general, 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. Thus, exploiting the power of the genetic algorithm (i.e., NSGA-II). GeNe finds more solutions, with a better quality.

GeNePi is by far the best algorithm, and I explain it by the composition of its elements: Ge (i.e., modified GRASP) finds a large number of solutions, allowing NSGA-II (the second step) 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.

Table 4.4 summarises the improvement of GeNePi in comparison to the second best algorithm on both hypervolume and number of non-dominated solutions. I show these results with and without the hybrid methods.

Table 4.4 shows that GeNePi significantly outperforms all non-hybrid algorithms with an increase of nearly 500% 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 51% in terms of hypervolume against these same algorithms.

Table 4.4 shows that GeNePi also outperforms hybrid metaheuristics on every single instance, both quantitatively (more than 211% non-dominated solutions on average) and qualitatively (more than 15% increase in hypervolume on average).

GeNe being the second best algorithm illustrates well the importance of having a local search phase at the end of the optimisation and not only a two step method. The last step of GeNePi (i.e., PLS) allows finding more non-dominated solutions, while at the same time slightly improving the hypervolume.
4.4 Evaluation of GeNePi Against (Meta)Heuristics

Table 4.4: Summary of the improvement (in percent) obtained using GeNePi on both number of non-dominated solutions and hypervolume when applied on the different instances. The table includes results with (w) and without (w/o) taking into account hybrid algorithms.

<table>
<thead>
<tr>
<th>Instance</th>
<th>% Improvement of GeNePi against 2nd best algorithm # Solutions</th>
<th>hypervolume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Improvement of GeNePi against 2nd best algorithm # Solutions</td>
<td>hypervolume</td>
</tr>
<tr>
<td></td>
<td>w/o hybrid</td>
<td>w hybrid</td>
</tr>
<tr>
<td>a_1_1</td>
<td>157.47</td>
<td>111.32</td>
</tr>
<tr>
<td>a_1_2</td>
<td>600.00</td>
<td>313.64</td>
</tr>
<tr>
<td>a_1_3</td>
<td>594.74</td>
<td>388.89</td>
</tr>
<tr>
<td>a_1_4</td>
<td>240.00</td>
<td>81.33</td>
</tr>
<tr>
<td>a_1_5</td>
<td>475.51</td>
<td>151.79</td>
</tr>
<tr>
<td>a_2_1</td>
<td>225.35</td>
<td>51.97</td>
</tr>
<tr>
<td>a_2_2</td>
<td>795.45</td>
<td>380.49</td>
</tr>
<tr>
<td>a_2_3</td>
<td>201.49</td>
<td>201.49</td>
</tr>
<tr>
<td>a_2_4</td>
<td>803.57</td>
<td>216.25</td>
</tr>
<tr>
<td>a_2_5</td>
<td>685.71</td>
<td>189.47</td>
</tr>
<tr>
<td>b_1</td>
<td>520.51</td>
<td>317.24</td>
</tr>
<tr>
<td>b_2</td>
<td>1204.35</td>
<td>222.58</td>
</tr>
<tr>
<td>b_3</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td>b_4</td>
<td>436.36</td>
<td>280.65</td>
</tr>
<tr>
<td>Average</td>
<td>499.32</td>
<td>211.22</td>
</tr>
</tbody>
</table>

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.

Table 4.5 shows the execution time of the studied algorithms on the differ-
4.4 Evaluation of GeNePi Against (Meta)Heuristics

tent instances to obtain the aforementioned results (i.e., shown in Tables 4.2 and 4.3). 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 I 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.

Table 4.5: Average execution times (s) of 10 runs of GeNePi and other evaluated algorithms on the different instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>a_1_1</th>
<th>a_1_2</th>
<th>a_1_3</th>
<th>a_1_4</th>
<th>a_1_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>2</td>
<td>3,106</td>
<td>441</td>
<td>309</td>
<td>332</td>
</tr>
<tr>
<td>Instance</td>
<td>a_2_1</td>
<td>a_2_2</td>
<td>a_2_3</td>
<td>a_2_4</td>
<td>a_2_5</td>
</tr>
<tr>
<td>Time (s)</td>
<td>3,905</td>
<td>600</td>
<td>695</td>
<td>342</td>
<td>347</td>
</tr>
<tr>
<td>Instance</td>
<td>b_1</td>
<td>b_2</td>
<td>b_3</td>
<td>b_4</td>
<td></td>
</tr>
<tr>
<td>Time (s)</td>
<td>14,991</td>
<td>10,028</td>
<td>39,596</td>
<td>63,535</td>
<td></td>
</tr>
</tbody>
</table>

To give the reader a sense of what happens during the optimisation of the different algorithms, I 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 time of this new solution in the x-axis and the new hypervolume of the solution set in y-axis). I especially want to see here the relative impacts of the three phases of GeNePi.

We see from Figure 4.3 that algorithms from the First Fit family are only making some improvement for a limited number of instances (i.e., a_1_1, a_1_5, and a_2_1). GRASP has a somewhat similar behaviour, but achieves a better improvement when it can make any.

Although PLS finds many solutions, they are somehow local and thus have a marginal impact on the hypervolume (with the exception of instances a_1_5, b_1, b_3, and b_4).

As it was expected, Ge brings a very good improvement at the beginning (in the first third of the execution time). However, Ge finds fewer solutions,
4.4 Evaluation of GeNePi Against (Meta)Heuristics

that bring only little improvement for the last 2 thirds of its execution time. This is mainly due to the absence of communication between its steps, and also to the lack of reactivity in the design of this metaheuristic.

Other algorithms which have the component Ge (i.e., GeNe and GeNePi) also 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.

The NSGA-II component in GrNe tries to do the same. However, it is penalised by the poor initial population obtained from the GRASP step. Thus taking a large time to reach a significant hypervolume.

In the same way as Ge, NSGA-II also plateaus eventually in most instances. This happens when the crossover operations cannot get new ‘interesting’ solutions, making mutation the only successful operation. Thus, it makes sense to have a local search as a third component (running for a short period of time such as in GeNePi), as it allows the refinement of the Pareto front when NSGA-II stagnates, and it also improves the hypervolume and the number of non-dominated solutions.

Figure 4.3 – Continued on next page
4.4 Evaluation of GeNePi Against (Meta)Heuristics

Figure 4.3 – Continued from previous page

Figure 4.3 – Continued on next page
4.4 Evaluation of GeNePi Against (Meta)Heuristics

Figure 4.3: Average improvement curves of 10 runs of the different algorithms on the different instances. Each point is a new solution (or a set of new solutions).
4.5 Hybridisation

We identified in the previous chapter 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 3.3. A solution to that challenge would be to combine CPLEX or CBLNS with a good metaheuristic 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), the idea is to 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 metaheuristic as a starting point for its optimisation.

A comparison between a large number of algorithms has already been performed in the previous section going from First Fit family techniques, metaheuristics, to hybrid metaheuristics and showed that GeNePi outperforms state-of-the-art algorithms on both quantity and quality of solutions.

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., Ge) in order to find initial solutions representing the search space. Therefore, I remove the Ge component from GeNePi and apply it only when the initial techniques fail at finding a large enough number of solutions to fill out an entire initial population and initiate the second step (i.e., Ne). It might also happen that I end up with more solutions at the end of CPLEX or CBLNS than needed in Ne. Therefore, I filter the solutions in this case and only keep the fittest and the most spread amongst them using the crowding metric [74]. By giving a population with good solutions representing the search space, I hope that it would improve Ne’s performance and quicken its convergence.

I 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 I achieve?) and on the execution time (how much is the increase in execution time?).
4.5 Hybridisation

4.5.1 Combining CPLEX and GeNePi

We saw in the previous chapter 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. Therefore, I 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. I 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.

I take the implementation of CPLEX and I give its results to GeNePi, with non-dominated solutions found using CPLEX used as the initial population as shown in Figure 4.4. It might happen that CPLEX does not find enough solutions to fill out an entire population (in my 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, I filter them by only keeping the fittest (the non-dominated ones) for a faster convergence and the most isolated amongst them (using the crowding measure [74])) 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., Ge).

In my implementation, GeNePi applies 10 generations of its second phase (i.e., Ne) 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, I use the same parameters as in the previous section.
Figure 4.4: Flowchart representation of the combination of CPLEX and GeNePi (i.e., CPLEX+GeNePi).
4.5 Hybridisation

4.5.2 Combining CBLNS and GeNePi

We saw in the previous chapter that CBLNS scales well to 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’ I 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 4.5, I 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 my CBLNS, I only collect one solution (i.e., the final one) after each run of CBLNS with a particular weight vector. Given that I configure CBLNS to run with a number of vectors always smaller than the population size, I never exceed the population size. Therefore, it is never needed to filter CBLNS’ resulting set of solutions. Instead, it is needed to add other solutions to fill out the initial population using the Ge component in GeNePi. In this case, Ge is always enabled in GeNePi and CBLNS results act always as ‘seeds’ to GeNePi’s optimisation.

![Flowchart representation of the combination of CBLNS and GeNePi (i.e., CBLNS+GeNePi).](image)

Figure 4.5: Flowchart representation of the combination of CBLNS and GeNePi (i.e., CBLNS+GeNePi).
4.5 Hybridisation

Similarly to the implementation of GeNePi when combined with CPLEX, I also use a population size of 20 individuals. I also provide the generated set of solutions from CBLNS and Ge to the second phase of GeNePi (i.e., Ne) as an initial population to evolve and optimise it in 10 generations. The non-dominated solutions that are found after running Ne are then given to the third and last step of GeNePi (i.e., Pi). 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.

4.5.3 Comparative Study

Tables 4.6 and 4.7 show respectively the results obtained on the modified ROADEF/EURO instances from $a_{1,1}$ to $b_{4}$ in terms of hypervolume (hyp), number of non-dominated solutions (#sol) and execution time (s). The symbol ‘x’ indicates that the algorithm could not run on the corresponding instance. Results are obtained using 10 runs of GeNePi alone (the average is taken), CPLEX alone, CBLNS alone, and my 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. Notice that reference points used to compute the hypervolume values are generated using the found solutions that have the worst objectives values, and that are different from those used in Section 4.4. However, they are the same for all the algorithms on every instance (to have comparable hypervolume results). This is because CPLEX and CBLNS are often aggressive on some objective while deteriorating the others at the same time, thus finding solutions with worse objective values than the used reference point. Therefore, the reference points considered at the beginning are no longer valid and have to be revised.

Table 4.6 confirms that GeNePi succeeds in improving the hypervolume and getting a large number of non-dominated solutions while keeping the execution time relatively low (Table 4.7).

We also notice that CPLEX based techniques (i.e., CPLEX and CPLEX+ GeNePi) cannot optimise large scale instances (i.e., $b_{2, 5}$, $b_{3}$ and $b_{4}$). However, CPLEX outperforms GeNePi in terms of hypervolume on 8 instances
Table 4.6: Average results of 10 runs 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. I put ‘x’ when the algorithm could not run on the instance.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Metric</th>
<th>Initial</th>
<th>GeNePi</th>
<th>CPLEX</th>
<th>CPLEX+GeNePi</th>
<th>CBLNS</th>
<th>CBLNS+GeNePi</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>hyp (e15)</td>
<td>6.16</td>
<td>9.71</td>
<td>8.82</td>
<td>9.74</td>
<td>6.76</td>
<td>9.16</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>225</td>
<td>3</td>
<td>30</td>
<td>4</td>
<td>124</td>
</tr>
<tr>
<td>a_1_2</td>
<td>hyp (e18)</td>
<td>4.75</td>
<td>5.69</td>
<td>5.93</td>
<td>6.17</td>
<td>4.93</td>
<td>5.51</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>160</td>
<td>8</td>
<td>42</td>
<td>4</td>
<td>193</td>
</tr>
<tr>
<td>a_1_3</td>
<td>hyp (e18)</td>
<td>6.57</td>
<td>7.22</td>
<td>9.01</td>
<td>9.06</td>
<td>6.83</td>
<td>7.24</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>132</td>
<td>5</td>
<td>21</td>
<td>6</td>
<td>129</td>
</tr>
<tr>
<td>a_1_4</td>
<td>hyp (e18)</td>
<td>8.10</td>
<td>9.19</td>
<td>10.28</td>
<td>10.89</td>
<td>8.47</td>
<td>8.96</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>136</td>
<td>6</td>
<td>35</td>
<td>3</td>
<td>200</td>
</tr>
<tr>
<td>a_1_5</td>
<td>hyp (e18)</td>
<td>2.42</td>
<td>3.15</td>
<td>3.94</td>
<td>4.04</td>
<td>2.67</td>
<td>3.11</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>283</td>
<td>4</td>
<td>54</td>
<td>4</td>
<td>266</td>
</tr>
<tr>
<td>a_2_1</td>
<td>hyp (e19)</td>
<td>4.57</td>
<td>5.93</td>
<td>7.95</td>
<td>8.17</td>
<td>6.80</td>
<td>7.26</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>231</td>
<td>5</td>
<td>39</td>
<td>6</td>
<td>380</td>
</tr>
<tr>
<td>a_2_2</td>
<td>hyp (e20)</td>
<td>3.33</td>
<td>4.36</td>
<td>4.76</td>
<td>5.01</td>
<td>5.13</td>
<td>3.38</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>197</td>
<td>7</td>
<td>42</td>
<td>3</td>
<td>195</td>
</tr>
<tr>
<td>a_2_3</td>
<td>hyp (e20)</td>
<td>2.55</td>
<td>3.69</td>
<td>4.93</td>
<td>5.13</td>
<td>2.58</td>
<td>3.73</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>202</td>
<td>2</td>
<td>32</td>
<td>2</td>
<td>242</td>
</tr>
<tr>
<td>a_2_4</td>
<td>hyp (e20)</td>
<td>3.21</td>
<td>6.19</td>
<td>5.84</td>
<td>6.68</td>
<td>6.13</td>
<td>9.79</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>253</td>
<td>3</td>
<td>43</td>
<td>3</td>
<td>429</td>
</tr>
<tr>
<td>a_2_5</td>
<td>hyp (e20)</td>
<td>4.96</td>
<td>5.81</td>
<td>5.63</td>
<td>5.86</td>
<td>6.50</td>
<td>7.30</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>220</td>
<td>2</td>
<td>23</td>
<td>5</td>
<td>414</td>
</tr>
<tr>
<td>b_1</td>
<td>hyp (e21)</td>
<td>8.20</td>
<td>8.74</td>
<td>10.7</td>
<td>10.88</td>
<td>8.31</td>
<td>8.53</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>244</td>
<td>2</td>
<td>29</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>b_2</td>
<td>hyp (e23)</td>
<td>1.43</td>
<td>3.29</td>
<td>x</td>
<td>x</td>
<td>3.35</td>
<td>3.36</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>300</td>
<td>x</td>
<td>x</td>
<td>6</td>
<td>461</td>
</tr>
<tr>
<td>b_3</td>
<td>hyp (e24)</td>
<td>2.33</td>
<td>2.41</td>
<td>x</td>
<td>x</td>
<td>2.45</td>
<td>3.76</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>162</td>
<td>x</td>
<td>x</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>b_4</td>
<td>hyp (e23)</td>
<td>3.65</td>
<td>4.21</td>
<td>x</td>
<td>x</td>
<td>8.32</td>
<td>8.34</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>118</td>
<td>x</td>
<td>x</td>
<td>6</td>
<td>41</td>
</tr>
</tbody>
</table>

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 solu-
Table 4.7: Execution time (s) of 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. The symbol ‘x’ indicates that the algorithm could not run on the corresponding instance. The average of 10 runs is reported for GeNePi, CPLEX+GeNePi, CBLNS and CBLNS+GeNePi.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Initial</th>
<th>GeNePi</th>
<th>CPLEX</th>
<th>CPLEX+GeNePi</th>
<th>CBLNS</th>
<th>CBLNS+GeNePi</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>0</td>
<td>2</td>
<td>0.18</td>
<td>0.38</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>a_1_2</td>
<td>0</td>
<td>518</td>
<td>303</td>
<td>319</td>
<td>3,600</td>
<td>4,605</td>
</tr>
<tr>
<td>a_1_3</td>
<td>0</td>
<td>441</td>
<td>151</td>
<td>167</td>
<td>3,600</td>
<td>4,375</td>
</tr>
<tr>
<td>a_1_4</td>
<td>0</td>
<td>309</td>
<td>138</td>
<td>158</td>
<td>3,600</td>
<td>4,184</td>
</tr>
<tr>
<td>a_1_5</td>
<td>0</td>
<td>332</td>
<td>21</td>
<td>34</td>
<td>3,600</td>
<td>4,271</td>
</tr>
<tr>
<td>a_2_1</td>
<td>0</td>
<td>284</td>
<td>159</td>
<td>166</td>
<td>7,200</td>
<td>8,786</td>
</tr>
<tr>
<td>a_2_2</td>
<td>0</td>
<td>600</td>
<td>2,580</td>
<td>2,694</td>
<td>7,200</td>
<td>8,315</td>
</tr>
<tr>
<td>a_2_3</td>
<td>0</td>
<td>695</td>
<td>71</td>
<td>154</td>
<td>7,200</td>
<td>8,471</td>
</tr>
<tr>
<td>a_2_4</td>
<td>0</td>
<td>342</td>
<td>20,655</td>
<td>21,173</td>
<td>36,000</td>
<td>36,558</td>
</tr>
<tr>
<td>a_2_5</td>
<td>0</td>
<td>347</td>
<td>22,513</td>
<td>22,548</td>
<td>36,000</td>
<td>36,591</td>
</tr>
<tr>
<td>b_1</td>
<td>0</td>
<td>14,991</td>
<td>8,913</td>
<td>9,452</td>
<td>36,000</td>
<td>43,225</td>
</tr>
<tr>
<td>b_2</td>
<td>0</td>
<td>10,028</td>
<td>x</td>
<td>x</td>
<td>36,000</td>
<td>44,249</td>
</tr>
<tr>
<td>b_3</td>
<td>0</td>
<td>39,596</td>
<td>x</td>
<td>x</td>
<td>36,000</td>
<td>43,221</td>
</tr>
<tr>
<td>b_4</td>
<td>0</td>
<td>63,535</td>
<td>x</td>
<td>x</td>
<td>36,000</td>
<td>43,392</td>
</tr>
</tbody>
</table>

...
4.5 Hybridisation

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. 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 a 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 the 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, I 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 CPLEX. CBLNS does not perform as well as GeNePi and CPLEX on small and medium instances, but gets better results when it comes to large scale instances. 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. I have a similar concluding remark for CBLNS with GeNePi: the hybridisation brings better results with a limited extra time required. The advantage of my approach is that it can be adapted to the size of the problem: when the size increases, I have the option to either continue relaxing some of the parameters used for CPLEX (i.e., fewer vectors or a larger optimality gap) or to replace CPLEX with CBLNS to feed in GeNePi.

4.6 Evaluation Against an Exact Resolution

In this section I compare the performance of GeNePi, CPLEX+GeNePi and CBLNS+GeNePi against an exact resolution algorithm. Several methods exist in the literature to solve exactly multi-objective problems with different complexities and outcome sets. Although all exact resolution methods obtain the optimal Pareto front, they diverge in the type of non-dominated solutions set they find. While some of them find the entire non-dominated solutions (e.g., the \( \varepsilon \)-Constraints method [62]), others only find the solutions with objective values at the convex hull of the optimal Pareto front (e.g., a weighted sum based method [60]).

In this chapter, I chose to compare the best algorithms against \( \varepsilon \)-Constraints method. This is motivated by two main reasons: (i) getting the entire Pareto front and not only solutions at the corners of the convex hull, and (ii) not having a need to aggregate objectives of different types and scales.

The \( \varepsilon \)-Constraints method is a good fit, as it finds the exhaustive Pareto front and it considers every objective separately and does not aggregate them in any way.

4.6.1 Description of the Epsilon Constraints Method

The \( \varepsilon \)-Constraints method is based on the transformation of a 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 \( \varepsilon \).
4.6 Evaluation Against an Exact Resolution

Let us assume the following multi-objective problem:

\[
\begin{align*}
\min & \quad (f_1(x), f_2(x), f_3(x)) \\
\text{s.t.} & \quad x \in \mathcal{X}.
\end{align*}
\] (4.1)

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 (4.1) into the following model:

\[
\begin{align*}
\min & \quad f_1(x) \\
\text{s.t.} & \quad f_2(x) \leq e_2 \\
& \quad f_3(x) \leq e_3 \\
& \quad x \in \mathcal{X}.
\end{align*}
\] (4.2)

\( \epsilon \)-Constraints method solves iteratively different instances of (4.2) using a succession of \( \mathcal{E} = \{e_2, e_3\} \) vectors, which probe the entire Pareto optimal front.

4.6.2 Implementation of the Epsilon Constraints Method

I have implemented the \( \epsilon \)-Constraints method on my problem considering the objectives mentioned in the previous chapter (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 \( \epsilon \) 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 \( \epsilon \) impossible without taking a risk of either not finding all the non-dominated solutions (if \( \epsilon \) is set to be too large) or having computational rounding errors (if \( \epsilon \) is set to be too small). That is why I chose to always keep the electricity cost as the main optimised objective, while generating constraints from the other objectives with variable \( \epsilon \) values.

In my implementation of the \( \epsilon \)-Constraints method, I exploit the linear aspect of my problem while solving iterative mono-objective problems. I 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
4.6 Evaluation Against an Exact Resolution

Figure 4.6: Execution time in number of days of the \( e \)-Constraints method on the used instances.

Execution environment (i.e., one node of a computing cluster with 24 cores 2.0GHz Intel \( \text{\textregistered} \) Ivy Bridge CPU and 128GB of RAM).

4.6.3 Results Obtained Using Epsilon Constraints Method

I run \( e \)-Constraints method on all instances used in the previous section, for a maximum of 30 days, and I extracted performance counters every 10 days. This means that the \( e \)-Constraints method running for a month on a powerful machine cannot always give the optimal answer. This probably makes clear how complex and large my problem is. We notice from Figure 4.6 that we only get optimal/complete Pareto front for four instances out of 14 during the 30 days (i.e., \( a_{1\_1}, a_{1\_5}, a_{2\_1}, b_{1} \)), with a total execution time of respectively 5.6, 3.25, 3.10 and 4.23 days. We could only get a partial optimal Pareto front for the rest of the instances during the 30 days of my experiment.

Tables 4.8 and 4.9 show respectively the results in terms of hypervolume and number of non-dominated solutions and execution time obtained after running the \( e \)-Constraints method on the different instances. The hypervolume is measured every 10 days for a duration of 30 days. Thus, getting a unique hypervolume and number of Pareto optimal solutions for the instances
which could be solved within the first 10 days (I put ‘–’ to indicate that the \( \varepsilon \)-Constraints method finished earlier) and three for the instances that could not be solved within the maximum execution time (i.e., 30 days). For comparison purposes, I also include a summary of the former results obtained with GeNePi, CPLEX+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, and I also put ‘x’ for missing results when CPLEX cannot handle certain instances) and CBLNS+GeNePi (using 5 vectors).

We see that results obtained after 10 days of execution time of \( \varepsilon \)-Constraints method 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 \( \varepsilon \)-Constraints method on the different instances at a given time. We also notice (as expected) that both the quantity of solutions and their respective quality keep increasing with the execution time, which indicates that \( \varepsilon \)-Constraints method 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. This makes the increase in the number of solutions in the Pareto front non-linear (for instance in a_1_2, 37, 17, and 24 new solutions are found respectively between 0 and 10 days, between 10 and 20 days, and between 20 and 30 days). It is also due to the fact that they are located close to each other, making the increase in hypervolume marginal (for instance in a_1_2, \( \varepsilon \)-Constraints method finds 17 new solutions between 10 and 20 days adding an improvement of 0.4e18 in hypervolume, and finds 24 new solutions between 20 and 30 days while only making an improvement of 0.05e18 in hypervolume).

We see from Table 4.8 that GeNePi is always outperformed by \( \varepsilon \)-Constraints method, but reaches a good ratio of its hypervolume while not requiring as much time. GeNePi reaches 68.29% of the hypervolume obtained by \( \varepsilon \)-Constraints method after running for 10 days. Moreover, GeNePi’s good performance wrt. \( \varepsilon \)-Constraints method does not decrease much over time:
### 4.6 Evaluation Against an Exact Resolution

Table 4.8: Average hypervolume and number of non-dominated solutions of 10 runs obtained with GeNePi, CPLEX+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+GeNePi using 5 vectors, and \( \varepsilon \)-Constraints method (\( \varepsilon \text{-Const} \)) run for 10, 20 and 30 days. I put ‘–’ when \( \varepsilon \text{-Const} \) finished earlier, and ‘\( x \)’ when CPLEX cannot handle the instance.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Metric</th>
<th>Initial</th>
<th>GeNePi</th>
<th>CPLEX+GeNePi</th>
<th>CBLNS+GeNePi</th>
<th>( \varepsilon \text{-Const} ) 10days</th>
<th>( \varepsilon \text{-Const} ) 30days</th>
<th>( \varepsilon \text{-Const} ) 30days</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>hyp (e15)</td>
<td>6.16</td>
<td>9.71</td>
<td>9.74</td>
<td>9.16</td>
<td>15.39</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>225</td>
<td>30</td>
<td>124</td>
<td>1280</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a_1_2</td>
<td>hyp (e18)</td>
<td>4.75</td>
<td>5.69</td>
<td>6.17</td>
<td>5.51</td>
<td>5.72</td>
<td>6.12</td>
<td>6.17</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>160</td>
<td>42</td>
<td>193</td>
<td>37</td>
<td>54</td>
<td>78</td>
</tr>
<tr>
<td>a_1_3</td>
<td>hyp (e18)</td>
<td>6.57</td>
<td>7.22</td>
<td>9.06</td>
<td>7.24</td>
<td>7.44</td>
<td>7.87</td>
<td>8.11</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>132</td>
<td>21</td>
<td>129</td>
<td>83</td>
<td>152</td>
<td>212</td>
</tr>
<tr>
<td>a_1_4</td>
<td>hyp (e18)</td>
<td>8.10</td>
<td>9.19</td>
<td>10.89</td>
<td>8.96</td>
<td>9.56</td>
<td>9.83</td>
<td>9.98</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>136</td>
<td>35</td>
<td>200</td>
<td>80</td>
<td>149</td>
<td>223</td>
</tr>
<tr>
<td>a_1_5</td>
<td>hyp (e18)</td>
<td>2.42</td>
<td>3.15</td>
<td>4.04</td>
<td>3.11</td>
<td>4.12</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>283</td>
<td>54</td>
<td>266</td>
<td>56</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a_2_1</td>
<td>hyp (e19)</td>
<td>4.57</td>
<td>5.93</td>
<td>8.17</td>
<td>7.26</td>
<td>10.25</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>231</td>
<td>39</td>
<td>380</td>
<td>109</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a_2_2</td>
<td>hyp (e20)</td>
<td>3.33</td>
<td>4.36</td>
<td>5.01</td>
<td>4.37</td>
<td>5.49</td>
<td>5.74</td>
<td>5.93</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>197</td>
<td>42</td>
<td>195</td>
<td>2994</td>
<td>4005</td>
<td>6603</td>
</tr>
<tr>
<td>a_2_3</td>
<td>hyp (e20)</td>
<td>2.55</td>
<td>3.69</td>
<td>5.13</td>
<td>3.73</td>
<td>3.86</td>
<td>4.06</td>
<td>4.21</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>202</td>
<td>32</td>
<td>242</td>
<td>2890</td>
<td>3421</td>
<td>4173</td>
</tr>
<tr>
<td>a_2_4</td>
<td>hyp (e20)</td>
<td>3.21</td>
<td>6.19</td>
<td>6.68</td>
<td>9.79</td>
<td>6.41</td>
<td>6.54</td>
<td>6.62</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>253</td>
<td>43</td>
<td>429</td>
<td>615</td>
<td>1034</td>
<td>1440</td>
</tr>
<tr>
<td>a_2_5</td>
<td>hyp (e20)</td>
<td>4.96</td>
<td>5.81</td>
<td>5.86</td>
<td>7.30</td>
<td>6.06</td>
<td>6.43</td>
<td>6.56</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>220</td>
<td>23</td>
<td>414</td>
<td>1355</td>
<td>2478</td>
<td>3516</td>
</tr>
<tr>
<td>b_1</td>
<td>hyp (e21)</td>
<td>8.20</td>
<td>8.74</td>
<td>10.88</td>
<td>8.53</td>
<td>11.03</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>244</td>
<td>29</td>
<td>30</td>
<td>272</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>b_2</td>
<td>hyp (e23)</td>
<td>1.43</td>
<td>3.29</td>
<td>x</td>
<td>3.36</td>
<td>3.39</td>
<td>3.52</td>
<td>3.57</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>300</td>
<td>x</td>
<td>461</td>
<td>31</td>
<td>61</td>
<td>89</td>
</tr>
<tr>
<td>b_3</td>
<td>hyp (e24)</td>
<td>2.33</td>
<td>2.41</td>
<td>x</td>
<td>3.76</td>
<td>2.47</td>
<td>2.49</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>162</td>
<td>x</td>
<td>30</td>
<td>246</td>
<td>478</td>
<td>714</td>
</tr>
<tr>
<td>b_4</td>
<td>hyp (e23)</td>
<td>3.65</td>
<td>4.21</td>
<td>x</td>
<td>8.34</td>
<td>4.25</td>
<td>4.33</td>
<td>4.35</td>
</tr>
<tr>
<td></td>
<td>#sol</td>
<td>1</td>
<td>118</td>
<td>x</td>
<td>41</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>
Table 4.9: Execution time (s) of GeNePi, CPLEX+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+GeNePi using 5 vectors (average of 10 runs for GeNePi, CPLEX+GeNePi and CBLNS+GeNePi), and $\varepsilon$-Constraints method ($\varepsilon$-Const) run for respectively 10, 20 and 30 days. I put ‘–’ to indicate that the $\varepsilon$-Constraints method finished earlier, and I also put ‘x’ for missing results when CPLEX cannot handle certain instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Initial</th>
<th>GeNePi</th>
<th>CPLEX+GeNePi</th>
<th>CBLNS+GeNePi</th>
<th>$\varepsilon$-Const 10days</th>
<th>$\varepsilon$-Const 30days</th>
<th>$\varepsilon$-Const 30days</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>0</td>
<td>2</td>
<td>0.38</td>
<td>31</td>
<td>483,747</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a_1_2</td>
<td>0</td>
<td>518</td>
<td>319</td>
<td>4,605</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>a_1_3</td>
<td>0</td>
<td>441</td>
<td>167</td>
<td>4,375</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>a_1_4</td>
<td>0</td>
<td>309</td>
<td>158</td>
<td>4,184</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>a_1_5</td>
<td>0</td>
<td>332</td>
<td>34</td>
<td>4,271</td>
<td>280,749</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a_2_1</td>
<td>0</td>
<td>284</td>
<td>166</td>
<td>8,786</td>
<td>267,638</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a_2_2</td>
<td>0</td>
<td>600</td>
<td>2,694</td>
<td>8,315</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>a_2_3</td>
<td>0</td>
<td>695</td>
<td>154</td>
<td>8,471</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>a_2_4</td>
<td>0</td>
<td>342</td>
<td>21,173</td>
<td>36,558</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>a_2_5</td>
<td>0</td>
<td>347</td>
<td>22,548</td>
<td>36,591</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>b_1</td>
<td>0</td>
<td>14,991</td>
<td>9,452</td>
<td>43,225</td>
<td>365,675</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>b_2</td>
<td>0</td>
<td>10,028</td>
<td>x</td>
<td>44,249</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>b_3</td>
<td>0</td>
<td>39,596</td>
<td>x</td>
<td>43,221</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
<tr>
<td>b_4</td>
<td>0</td>
<td>63,535</td>
<td>x</td>
<td>43,392</td>
<td>864,000</td>
<td>1,728,000</td>
<td>2,592,000</td>
</tr>
</tbody>
</table>

GeNePi still gets 61.92% after 20 days and 59.55% after 30 days of running $\varepsilon$-Constraints method. We also see that CPLEX+GeNePi does not optimise the large scale instances $b_2$, $b_3$ and $b_4$, but when it does, CPLEX+GeNePi reaches 113.75%, 102.57% and 97.52% of the hypervolume obtained by $\varepsilon$-Constraints method run respectively for 10, 20 and 30 on average. The combination of CPLEX and GeNePi achieves even better results in terms of hypervolume than $\varepsilon$-Constraints method run for 30 days in 4 instances out of 11. CBLNS+GeNePi also achieves good results on most of the instances in terms of hypervolume and reaches 158.98%, 141.61% and 136.73% of the hypervol-
4.6 Evaluation Against an Exact Resolution

ume obtained by $\varepsilon$-Constraints method run respectively for 10, 20 and 30 on average. Furthermore, CBLNS+GeNePi outperforms $\varepsilon$-Constraints method run for 30 days in 5 instances out of 14. However, despite having a good overall performance in comparison to $\varepsilon$-Constraints method, CBLNS+GeNePi does not achieve as good results as CPLEX+GeNePi when this one could optimise them (CPLEX+GeNePi achieves better results than CBLNS+GeNePi on 9 out of 11 instances).

Table 4.8 shows that GeNePi outperforms $\varepsilon$-Constraints method in terms of number of non-dominated solutions with respectively more than 360% (10 days), 238% (20 days) and 186% (30 days) on average. However, if we take a look at the different instances in more details, we observe that GeNePi does not always outperform $\varepsilon$-Constraints method (e.g., GeNePi does poorly on $a_2-2$ and $a_2-3$). CBLNS+GeNePi also finds a decent amount of non-dominated solutions with respectively more than 300%, 211% and 167% of $\varepsilon$-Constraints method run for 10, 20 and 30 days on average. However, we see that CPLEX+GeNePi does not perform that well on this metric as it only reaches 24.21% (10 days), 19.10% (20 days) and 16.41% (30 days) of number of non-dominated solutions found by $\varepsilon$-Constraints method.

We see from Table 4.9 that GeNePi is 1,000 times faster on average than $\varepsilon$-Constraints method, without taking into account 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 $\varepsilon$-Constraints method with at least one order of magnitude. It is even faster than a single solution found by $\varepsilon$-Constraints method 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 $\varepsilon$-Constraints method) with one of the best commercial MILP solvers (i.e., CPLEX) while running on a cluster node. We also notice that $\varepsilon$-Constraints method average execution time per solution increases over time (for instance on $a_1-2$, it goes from 23,351s in 10 days to 32,000s in 20 days, and 33,231s in 30 days). This consolidates the aforementioned result that the iterative mono-objective problems get harder to solve over time. CPLEX+GeNePi and CBLNS+GeNePi with the chosen parameters are also always faster than $\varepsilon$-Constraints method running for 10 days. We see that CPLEX+GeNePi is
even quicker than GeNePi as it is almost 3,000 times faster on average than $\epsilon$-Constraints method (excluding instance $a_{1\_1}$). However, this is without including CPLEX+GeNePi’s results with large scale instances, as I expect this number to decrease after adding them. CBLNS+GeNePi is also fast, but not as much as GeNePi and CPLEX+GeNePi as CBLNS+GeNePi is nearly 100 times faster on average than $\epsilon$-Constraints method (excluding instance $a_{1\_1}$).

4.7 Conclusion

In this chapter, I compared 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.

I showed on a large experimental validation that GeNePi outperforms other non-hybrid algorithms: it finds nearly 5 times more good solutions (non-dominated) that are scattered over more of the search space (hypervolume is more than 100% better) – which is desirable as I want to offer decision makers a large variety of different solutions. GeNePi also outperforms other hybrid metaheuristics (it finds more than 2 times more non-dominated solutions and achieves a better hypervolume with over 15% on average).

I also proposed to feed the results of CPLEX and CBLNS to a metaheuristic (GeNePi) and I compared CPLEX alone, CBLNS alone, GeNePi alone, CPLEX+GeNePi and CBLNS+GeNePi. We observed 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 my benchmark). On the other hand, GeNePi and CBLNS scale to large instances, but 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.

A comparison of my algorithm against one of the well-known exact methods for solving multi-objective problems (i.e., \(\varepsilon\)-Constraints method) shows that GeNePi gets more than 186% non-dominated solutions and a hypervolume of more than 59% on average than \(\varepsilon\)-Constraints method 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 times faster on average than \(\varepsilon\)-Constraints method, and even faster or on the same order of magnitude as one single mono-objective optimisation of the same problem. CPLEX+GeNePi does not optimise all the instances, but achieves good results when it does. CPLEX+GeNePi reaches more than 97% of the hypervolume obtained by \(\varepsilon\)-Constraints method run for 30 days, but only finds 16.41% of its non-dominated solutions on average. CBLNS+GeNePi achieves worse results in terms of hypervolume than CPLEX+GeNePi, but optimises all the instances while reaching more than 136% of hypervolume obtained by \(\varepsilon\)-Constraints method after running for 30 days and finding more than 167% of its non-dominated solutions on average.

In the next chapter, I include the decentralisation aspect to the large data centres and study the multi-objective VM reassignment in decentralised data centres, which are composed of several autonomous hosting departments with their own preferences and views on how to make their infrastructure better.
In this chapter, I formally define the problem of multi-objective VM reassignment for large decentralised data centres. I also propose E-GeNePi, a solution that uses a multi-layer architecture and an adaptation of GeNePi (my solution from Chapter 4) to suggest reassignment solutions that are evaluated by the various hosting departments according to their preferences.

5.1 Introduction

Large modern organisations often face internal segmentation, based on geography, legal jurisdictions, age/maturity of lines of business, and sheer size. This process, sometimes referred to as siloing, means that the organisation has employees who identify themselves with their group rather than with the whole organisation, and who view their group’s objectives as being more relevant than the organisation’s objectives. Capital allocators of the different hosting departments sometimes compete or at least have different perspectives on the best way of making the system better. Figure 5.1 shows an example of such data centres that I call decentralised data centres, which is composed of three VCs.
5.1 Introduction

Hence the research challenge I address in this chapter is the possibility to build a system that: (i) satisfies individual CAs’ placement preferences, (ii) offers top-level decision makers with a panel of possible and validated good assignments that they can navigate and (iii) scales up to large data centres.

In this chapter, I use the phrase data centre for the global IT infrastructure of a large company, while I use virtual data centre (VC), or hosting site/department, for each smaller group of servers and VMs that make sense by themselves. For instance, a company newly acquired by a bigger one and keeping its servers to host its workloads would be considered as a VC. Likewise, a particular department in a big group ‘possessing’ its own IT infrastructure would be considered a VC. I describe as decentralised the type of data centres that my system targets, as they are composed of several VCs with a certain degree of autonomy in the placements they favour.

I also use ‘objectives’ and ‘preferences’ in this chapter and I would like to clarify the distinction. The objectives are the different dimensions that need to be optimised. For instance, I define in the next section three objectives that make sense for practitioners when optimising a data centre (see next sections for more details): electricity consumption, system reliability and migration. These objectives are agnostic towards any particular goal or target: they just define the multi-objective search space of all possible assignments. The pref-
5.2 Formal Problem Definition

I propose in this section the first full formalisation of the multi-objective VM reassignment problem in large decentralised data centres. I present the variables defining a data centre: physical machines, virtual machines, individual virtual data centres; as well as the constraints, the cost functions and the optimisation problems: placement and reassignment.

A data centre is composed of a set $\mathcal{C}$ of VCs: $\mathcal{C} = \{c_1, \ldots, c_n\}$. Each $c_i$ is given a set $\mathcal{M}_i \subseteq \mathcal{M}$ of PMs, $\mathcal{M}_i = \{m_1, \ldots, m_n\}$. Each $m_j$ has several resources $r \in \mathcal{R}$ (e.g., CPU, RAM and disk), in limited capacity $Q_{m,j,r}$. Resources are either transient ($r \in \mathcal{T}\mathcal{R} \subseteq \mathcal{R}$, such as RAM and disk) if they are consumed by both origin and destination PMs during a VM migration, or non-transient otherwise (e.g., CPU): $r \in \mathcal{R} \setminus \mathcal{T}\mathcal{R}$. The data centre hosts a set $\mathcal{V}$ of VMs, $\mathcal{V} = \{v_1, \ldots, v_l\}$: $M(v_k) = m_j$ is used to denote the PM on which $v_k$ is hosted (I also use the notation $M_0(v_k) = m_j$ for the original PM of $v_k$ in case of a migration). The quantity of resource $r$ that every $v_k$ needs is fixed to $d_{k,r}$. VMs are sometimes grouped into services $\mathcal{S} = \{s_1, \ldots, s_p\}$, with $s_p = \{v_{p1}^1, \ldots, v_{p1}^{d_p}\}$. A service can be seen as a distributed application, such as one implementing a multi-tier architecture.
5.2 Formal Problem Definition

5.2.1 Constraints of the Problem

Every placement of VM on a PM is subject to a set of constraints that I describe below.

5.2.1.1 Capacity Constraints

It is required that VMs do not exceed the resource capacity of the PM they are hosted on or they are migrating to/from (in case of transient resources), \( \forall m \in M \):

\[
\begin{align*}
\sum_{v_k \in V \mid M(v_k) = m \land d_{v_k,r} \leq Q_{m,r}} & \leq Q_{m,r} \quad \text{if } r \in \mathcal{T} \mathcal{R} \\
\sum_{v_k \in V \mid M(v_k) = m \land d_{v_k,r} > Q_{m,r}} & \leq Q_{m,r} \quad \text{otherwise}
\end{align*}
\]

(5.1)

5.2.1.2 Conflict Constraints

Services running on several VMs may prefer to use a different PMs.

\[
\forall v_k, v_l \in V, k \neq l, \forall s_p \in S, (v_k, v_l) \in s_p^2 \Rightarrow M(v_k) \neq M(v_l)
\]

(5.2)

Network connections between PMs in data centres vary and this can have an impact on CAs’ decisions on the location of VMs. The concept of neighbourhood is used to capture this and I denote it \( N(m_i) = \{ m_i^1, \ldots, m_i^k \} \) for a PM \( m_i \).

5.2.1.3 Dependency Constraints

If a service \( s_i \) depends on \( s_j \) (\( s_i \leftarrow s_j \)), then every VM \( v_k \) in \( s_i \) should be in the neighbourhood of at least one VM \( v_l \) from \( s_j \):

\[
s_i \leftarrow s_j \implies \forall v_k \in s_i, \exists v_l \in s_j \mid N(M(v_k)) = N(M(v_l))
\]

(5.3)
5.2 Formal Problem Definition

5.2.1.4 Spread Constraints

Some services require their VMs to be allocated to different VCs, for instance for security reasons:

$$\sum_{c_i \in C} \min \{1, |\{v \mid v \in s \land VC(v) = c_i\}|\} \geq \text{spread}_s, \ \forall s \in S \quad (5.4)$$

where \(\text{spread}_s\) is the minimum number of VCs that have to contain at least one VM from the service \(s\), and \(VC(v)\) is the VC where the VM \(v\) is located.

5.2.2 Objectives to Optimise

I now introduce the cost functions defining the different objectives.

5.2.2.1 Reliability Cost

A PM \(m_j\) is said reliable if its load for every resource \(r \in R\) does not exceed a safety capacity \(\rho(m_j, r)\), and I define the reliability cost associated as:

$$\rho(m_j) = \sum_{r \in R} \max \left(0, Q_{m_j, r} - \rho(m_j, r)\right) \quad (5.5)$$

5.2.2.2 Electricity Cost

The electricity cost of a PM \(m_j \in M\) is defined by the electricity price (per unit) at \(m_j\’s\) location multiplied by \(m_j\’s\) electricity consumption. Modelling electricity consumption is complex and I use here some simplified model [171, 184] considering that it is a linear function of the CPU usage:

$$\epsilon(m_j) = \begin{cases} \gamma_{m_j} \times \left(\alpha_{m_j} \times Q_{m_j, \text{CPU}} + \beta_{m_j}\right) & \text{if } m_j \text{ is running} \\ 0 & \text{otherwise} \end{cases} \quad (5.6)$$

where \(\gamma_{m_j}\) is the electricity price at the location of \(m_j\), \(\beta_{m_j}\) its electricity consumption when idle, and \(\alpha_{m_j}\) the tangent of the CPU consumption.
5.3 E-GeNePi

5.2.2.3 Migration Cost

The cost of migrating a VM \( v \) is the composition of three costs [212]: preparation of the migration: \( \mu_1(v, M_0(v)) \), transmission: \( \mu_2(v, M_0(v), M(v)) \) and installation in the new host: \( \mu_3(v, M(v)) \). Note that each of these costs depends on various parameters, such as VM size and network topology.

\[
\mu(v, M_0(v), M(v)) = \mu_1(v, M_0(v)) + \mu_2(v, M_0(v), M(v)) + \mu_3(v, M(v)) \quad (5.7)
\]

Definition 2 (Placement) Given a VC \( c_i \), its PMs \( M_i \) and a set of VMs \( V_i = \{v_1, \ldots, v_2\} \), a placement of \( V_i \) on \( M_i \) is a mapping: \( \text{Plac}_i : V_i \mapsto M_i \), such that \( \text{Plac}_i(v) \rightarrow m \), which satisfies the constraints 5.1, 5.2, 5.3.

The preference \( \text{Pref}(c_i) \) of each VC \( c_i \) corresponds to a function to minimise and is defined by three constants \( w_1^i, w_2^i \) and \( w_3^i \):

\[
\text{Pref}(c_i) = w_1^i \times \sum_{m_j \in M_i} \rho(m_j) + w_2^i \times \sum_{v_k \in V_i} \mu(v_k, M_0(v_k), \text{Plac}(v, M_i)) + w_3^i \times \sum_{m_j \in M_i} \epsilon(m_j) \quad (5.8)
\]

Definition 3 (Reassignment) A reassignment \( \text{ReA} \) of VMs to VCs is a mapping: \( \text{ReA} : V \mapsto C \), such that \( \text{ReA}(v_k) \rightarrow c_i \), which satisfies the constraint 5.4 and the constraints 5.1, 5.2 and 5.3 at \( c_i \)'s level.

As this is a multi-objective context, I do not try to minimise one single objective function but rather all of the objectives (defined as minimisation of the various cost functions 5.5, 5.7 and 5.6) at the same time. I am searching for the set of solutions that are better than any other on a particular combination of objectives, and are not comparable to any other on the Pareto front.

5.3 E-GeNePi

E-GeNePi is my multi-objective VM reassignment system for large decentralised distributed data centres. The goal of E-GeNePi is to satisfy the preferences of the individual VC’s CAs while offering the top managers with better
placement solutions in the multi-objective search space. E-GeNePi is composed of two different modules (see Figure 5.2):

### 5.3.1 Reassignment

The decision makers at the organisation level run an adaptation of GeNePi (my three-step metaheuristic defined in Chapter 4). Decision makers send every reassignment solution found by GeNePi to the VCs, where a placement is tried: if a placement is possible for every VC, then the real cost of the solution is updated (and the solution possibly discarded if not good enough); otherwise GeNePi modifies the solution and resubmits it. The first step of GeNePi corresponds to the constructive phase of GRASP [115]: VMs are sorted according to their dependencies and requirements (decreasingly) and the top ones are assigned iteratively to VCs that respect services spread, allow the satisfaction of dependency constraints and have a utility higher than a given threshold $a$. GeNePi then applies a specific genetic algorithm NSGA-II [74] where: (i) four parents are selected instead of two during the evolution phase.
and a tournament is organised by pairs, before mixing the winners (using crossover and mutation operators); (ii) the evolved population is mixed with the original one in order to only keep the elites and quicken the convergence towards non-dominated solutions.

To be more precise, the crossover in my system is performed by one or two ‘cuts’ at random positions in a solution’s parents chromosomes and the resulting parts are interchanged, while the mutation consists of reassigning a random VM to a random VC.

Eventually GeNePi uses a local search, PLS [218], which looks for new solutions in the neighbourhood of the most isolated non-dominated solutions (as they are the most likely to refine the non-dominated set). GeNePi uses two operators in the generation of the neighbourhood: 1-exchange and swap.

I define an execution time for each step instead of fixing the number of iterations. This allows to have more control over the global execution time and to adapt to the complexity of the instances (e.g., an iteration of GRASP is faster when the constructed solution in infeasible). GRASP is given a third of the execution time, and that can be extended up to the half if the size of the initial population for NSGA-II is not reached. NSGA-II gets half of the global time, because of the large improvement it can achieve. PLS gets a sixth of the global time as it is only used to refine the non-dominated set. Note that I define the execution time of each step to a value that seems adequate, however without performing an extensive tuning analysis that could allow achieving a better performance.

5.3.2 Placement

Every VC runs two algorithms: (i) a placement algorithm to match allocated VMs to hosted PMs; and (ii) a Hill Climbing algorithm to optimise the placement. If the two algorithms are successful (i.e., the output is a feasible solution), CAs send the cost of placing the set of allocated VMs back to the decision makers. Otherwise, if no placement is possible, the VMs are sent back to the managers of the organisation. The placement process has to be done in less than a certain time (the same for all the VCs).
The first step is described in Algorithm 5.1 and consists of finding an initial placement satisfying every VC’s constraints. This is done by placing VMs already in the VC at their initial PM, thus minimising the migration cost. The rest of the VMs are sorted by dependencies and requirements, and placed one after the other using a First Fit Decreasing. The repair mechanism deals with VMs that cannot find a placement and moves at most two other VMs, or cancels the placement.

**Algorithm 5.1: VC’s initial VM placement**

**input**: \( M_i; PMs, \; V_i; VMs \)

**output**: \( V_{not\_placed}; VMs \)

1. \( V_{not\_placed} \leftarrow \emptyset; \; V_{placed} \leftarrow \emptyset; \)
   
   // VC’s initial VMs \( V_{init} \) are not moved

2. \( V_{new} \leftarrow V\backslash V_{init}; \)
   
   // Placing the rest of VMs

3. \( sort(V_{new}, \; Dependency \; and \; Requirement); \)

4. for \( v \in V_{new} \) do

5.     if exists( FFD(\( v \)) ) then

6.         \( M(v) \leftarrow FFD(\( v \)); \; V_{placed} \leftarrow V_{placed} \cup \{v\}; \)

7.     else

8.         \( V_{not\_placed} \leftarrow V_{not\_placed} \cup \{v\}; \)

   // Repair

9. for \( v \in V_{not\_placed} \) do

10.     if repaired(\( v, M_i, V_{placed} \)) then

11.         \( V_{not\_placed} \leftarrow V_{not\_placed} \backslash \{v\}; \)

12. return \( V_{not\_placed}; \)

The second step is summarised in Algorithm 5.2: it uses a Hill Climbing algorithm with two possible terminations: either when there is no more improvement in the objective function or when the time-out (set by the system) is reached. Similarly to PLS for the assignment, the idea is to find a better placement by exploring the possible solutions around the current feasible one,
5.4 Experimental Setup

using 1-exchange (which changes the placement of one single VM) and swap (which exchanges the placement of two VMs).

Algorithm 5.2: Optimisation of VC’s initial VM placement

<table>
<thead>
<tr>
<th>input</th>
<th>p: Placement, W: Weights, M: Moves, t: Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>p: Placement</td>
</tr>
</tbody>
</table>

1 do
2 \( \mathcal{N} \leftarrow \text{generateNeighbours}(p, M); \)
3 \((p', \text{saving}) \leftarrow \text{getBestSaving}(p, W, \mathcal{N}); \)
4 if \( \text{saving} > 0 \) then
5 \( p \leftarrow p' \)
6 while \( \text{saving} > 0 \) & \( \text{execTime} < t; \)
7 return \( p; \)

5.4 Experimental Setup

I would like in this section to evaluate how E-GeNePi performs against state-of-the-art multi-objective placement metaheuristics. I use the large dataset provided by Google to the ROADEF/EURO challenge. However, to adapt it to my context (decentralised DCs), I consider that every location in the original dataset is an independent VC in my dataset. I also randomly generate the objective preference weights for each of the VCs. I focus on 14 instances (see Table 5.1): all the instances of the sets a_1 (easy), 4 (out of 5) of the set a_2 (medium), and the first 5 (out of 10) of the set b (hard). I picked these instances only to limit the execution time of my experiments (I have 10 runs per algorithm and per instance). Notice that I do not include the instance a_2_1 (used in the previous chapter) as it only contains one location (i.e., one VC) thus only having one unique solution (i.e., the initial one). To keep the same number of instances, I substitute a_2_1 with a new one (i.e., b_5).

Instances from the set a_1 have between 2 and 4 resources, 4 to 100 PMs, 100 to 1,000 VMs, 79 to 981 services and 4 to 50 VCs; those from the set a_2 have 12 resources, 50 or 100 PMs, 1,000 VMs, 129 to 180 services and 25
Table 5.1: Instances size and allowed execution time.

<table>
<thead>
<tr>
<th>Instance</th>
<th># Resources</th>
<th># PMs</th>
<th># VMs</th>
<th># Services</th>
<th># VCs</th>
<th>Exec Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>2</td>
<td>4</td>
<td>100</td>
<td>79</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>a_1_2</td>
<td>4</td>
<td>100</td>
<td>1,000</td>
<td>980</td>
<td>4</td>
<td>1,200</td>
</tr>
<tr>
<td>a_1_3</td>
<td>3</td>
<td>100</td>
<td>1,000</td>
<td>216</td>
<td>25</td>
<td>1,200</td>
</tr>
<tr>
<td>a_1_4</td>
<td>3</td>
<td>50</td>
<td>1,000</td>
<td>142</td>
<td>50</td>
<td>1,200</td>
</tr>
<tr>
<td>a_1_5</td>
<td>4</td>
<td>12</td>
<td>1,000</td>
<td>981</td>
<td>4</td>
<td>1,200</td>
</tr>
<tr>
<td>a_2_2</td>
<td>12</td>
<td>100</td>
<td>1,000</td>
<td>170</td>
<td>25</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_3</td>
<td>12</td>
<td>100</td>
<td>1,000</td>
<td>129</td>
<td>25</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_4</td>
<td>12</td>
<td>50</td>
<td>1,000</td>
<td>180</td>
<td>25</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_5</td>
<td>12</td>
<td>50</td>
<td>1,000</td>
<td>153</td>
<td>25</td>
<td>3,600</td>
</tr>
<tr>
<td>b_1</td>
<td>12</td>
<td>100</td>
<td>5,000</td>
<td>2,512</td>
<td>10</td>
<td>7,200</td>
</tr>
<tr>
<td>b_2</td>
<td>12</td>
<td>100</td>
<td>5,000</td>
<td>2,462</td>
<td>10</td>
<td>7,200</td>
</tr>
<tr>
<td>b_3</td>
<td>6</td>
<td>100</td>
<td>20,000</td>
<td>15,025</td>
<td>10</td>
<td>7,200</td>
</tr>
<tr>
<td>b_4</td>
<td>6</td>
<td>500</td>
<td>20,000</td>
<td>1,732</td>
<td>50</td>
<td>7,200</td>
</tr>
<tr>
<td>b_5</td>
<td>6</td>
<td>500</td>
<td>40,000</td>
<td>35,082</td>
<td>10</td>
<td>7,200</td>
</tr>
</tbody>
</table>

VCs; while those from the set b have 6 or 12 resources, 100 or 500 PMs, 5,000 to 40,000 VMs, 1,732 to 35,082 services and 10 to 50 VCs. For example, the instance a_1_2 that I use in the next section to showcase the behaviour of the algorithms has 4 resources, 100 PMS, 1,000 VMs in 980 services and 4 VCs. E-GeNePi is allowed only a fixed execution time (and I force all other algorithms to give an answer in this limited time): 15 seconds for a_1_1, 1,200 seconds for the other instances of the set a_1, 3,600 seconds for the instances of the set a_2 and 7,200 seconds for the instances of the set b. These time limits are considered realistic in the context of my work (optimisation performed on a regular basis, e.g., monthly or quarterly).

I also use for my evaluation of the algorithms the same two metrics as in Chapter 3: the number of non-dominated solutions as a measure of the quantity of solutions found, and the hypervolume as a measure of the quality of the solutions (how much of the solution space they explore).

All my algorithms are implemented in C++ and the tests are performed on a machine running Ubuntu 12.4 LTS 64bits with 62GB of RAM and 24 Intel ©
5.5 Evaluation

Xeon ® 2.20GHz CPUs.

5.4.1 Other Algorithms

In absence of a state-of-the-art for my problem, I follow the same strategy as in the previous chapter and compare E-GeNePi against one algorithm for each of the most common metaheuristics (i.e., greedy, genetic algorithm and local search). I compare my solution E-GeNePi to five other algorithms that apply the same process at the VC level (find an initial placement and apply a Hill Climbing algorithm). They differ from each other on the multi-objective algorithm that generates the assignments (at the capital allocator level). In my experiments, I simplify the notation and distinguish the different algorithms by the name of the technique running at this level instead of using the name of the entire system (e.g., GeNePi instead of E-GeNePi). The first algorithm is E-GRASP, where GeNePi is replaced by GRASP. The second algorithm is E-NSGA-II that runs NSGA-II. The third algorithm (E-GrGA) has a two steps method with GRASP (one-third of the execution time) followed by NSGA-II (two-thirds of the execution time). E-PLS is the fourth algorithm, where the decision maker runs PLS during the entire execution, but applied on all the non-dominated solutions at every step. The last algorithm is E-HC: it runs several iterations of the Hill Climbing algorithm with different weight triplets (one weight per objective).

5.5 Evaluation

The main goal of this section is to compare the quantity and the quality of the solutions found by E-GeNePi against the other algorithms that I presented above. I am also interested in evaluating the Hill Climbing algorithms at the VCs level, as it is crucial for E-GeNePi to know whether the VCs find a placement for the VMs assigned to them in a reasonable time (i.e., the quicker the better). Finally, I am interested to know how the different algorithms work together in E-GeNePi, and I profile the resolution of one of the instances to understand the behaviour of the algorithms.
5.5 Evaluation

5.5.1 Evaluation of the VCs’ Hill Climbing

At the VC level (see Figure 5.2, Section 5.3) I use both a First Fit algorithm for the initial placement of VMs and a Hill Climbing to improve the placement. To understand how the Hill Climbing algorithm improves the placement, and how much time is needed to reach a stable state (when the algorithm converges), I show in Figure 5.3 the iterative savings of the instances with a sufficient number of PMs per VC (i.e., those where number of PMs $\gg$ number of VCs) – I use the very first assignment found by E-GeNePi here. I first show all the details of the process for the 4 VCs of instance $a_{1_2}$, each composed of 25 PMs and 225, 228, 284 and 233 VMs respectively. The first thing to say is that we do not see here the first placement (First Fit Decreasing step) but only the improvements of the placements allowed by the Hill Climbing. We notice that the savings are significant and decreasing.

Figure 5.3 – Continued on next page
5.5 Evaluation

Figure 5.3 – Continued from previous page

Figure 5.3 – Continued on next page
5.5 Evaluation

Figure 5.3: Placement savings obtained by the Hill Climbing algorithm. Savings are detailed for a\_1\_2 (4 VCs), and for the others, I show the area defined by the fastest and slowest VCs (in terms of convergence), and mean of the savings.

This is interesting as it means that CAs (or decision makers) can decide to allocate a shorter time for the improvement of the placement without impacting too much the quality of the process. Figure 5.3 also shows that for a\_1\_2 the placements quickly converge (sometimes locally) to an optimum: in this example all the placements converge before the end of the allocated time (4 seconds) – which may not always be the case. We also see that the original situation varies a lot and the amount of savings as well: during about the same execution time (3.64 and 2.95 seconds respectively) savings for c\_2 and c\_0 vary a lot (6,081,944 for c\_2 while 2,192,652 for c\_0). Likewise, c\_2 needs more time to converge than the other VCs. Both remarks are caused by the higher number of VMs on c\_2 than other VCs which makes improving the placement more complex (longer) while it exhibits more important savings.

Figure 5.3 also shows a simplified version of the savings for each of the other instances. Instead of showing the savings for every instance, and because most instances have a lot of VCs (e.g., 10 to 50 in most cases), I present only the VCs that converge the fastest (i.e., the first to have no improvement between any two iterations) and the slowest, and the mean of all VCs’ savings values. We observe that the convergence is very quick, less or around 1 second for all the a’s (but a\_1\_2), and in the order of the tens of seconds for the b’s. We also see that the difference between fastest and slowest VCs
is not huge, and anyway in the same order of magnitude (19.5s-60.42s for \( b_1 \) and 5.5s-19.93s for \( b_4 \) are the largest differences between fastest and slowest converging VCs). The mean values (dark lines in the grey areas) show that the converging time of the majority of the VCs tends to be close to the one of the slowest. It is not a problem as such, given that processing the VCs is likely to be done in parallel, i.e., at each VC’s and that even for the slowest the placement improvements converge in a reasonable time.

### 5.5.2 Quantity and Quality of Solutions

Tables 5.2 and 5.3 show the results obtained on the modified ROADEF/EURO instances in terms of hypervolume and number of non-dominated solutions.

Table 5.2: Summary of average solutions of 10 runs for the various algorithms and the various instances. The higher the better. I put in bold the best values for each instance, and in italics the second best.

<table>
<thead>
<tr>
<th>Instance</th>
<th>E-GeNePi</th>
<th>E-GrGA</th>
<th>E-GRASP</th>
<th>E-NSGA</th>
<th>E-PLS</th>
<th>E-HC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>416</td>
<td>15</td>
<td>25</td>
<td>20</td>
<td>6</td>
<td>276</td>
</tr>
<tr>
<td>a_1_2</td>
<td>36</td>
<td>26</td>
<td>22</td>
<td>37</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>a_1_3</td>
<td>74</td>
<td>40</td>
<td>23</td>
<td>44</td>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>a_1_4</td>
<td>192</td>
<td>163</td>
<td>11</td>
<td>189</td>
<td>42</td>
<td>72</td>
</tr>
<tr>
<td>a_1_5</td>
<td>39</td>
<td>34</td>
<td>17</td>
<td>20</td>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>a_2_2</td>
<td>62</td>
<td>46</td>
<td>23</td>
<td>55</td>
<td>6</td>
<td>65</td>
</tr>
<tr>
<td>a_2_3</td>
<td>74</td>
<td>39</td>
<td>18</td>
<td>72</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>a_2_4</td>
<td>118</td>
<td>92</td>
<td>27</td>
<td>113</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>a_2_5</td>
<td>67</td>
<td>95</td>
<td>19</td>
<td>59</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>b_1</td>
<td>17</td>
<td>14</td>
<td>18</td>
<td>18</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>b_2</td>
<td>17</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>b_3</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>b_4</td>
<td>16</td>
<td>6</td>
<td>8</td>
<td>11</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>b_5</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 5.3 shows that using E-GeNePi gives better results in terms of hypervolume (quality of solutions sets) for all instances, with an average of +6.03% improvement compared to the second best results. This means that E-GeNePi succeeds in finding new areas of the search space and exploring them. While E-GrGA, E-GRASP and E-NSGA give good results on average, E-NSGA is bad when the instances are big. This can be explained by the fact that NSGA is dependent on several initial solutions of good quality – and we have only one here (the initial placement), requiring to generate some random (and of poor quality) ones. Algorithms based on a local search (i.e., E-PLS and E-HC) have the worst results, which is due to the time it takes to evaluate an entire neighbourhood in absence of utility functions (the algorithms need to ask the VCs).

Table 5.3: Summary of average hypervolume of 10 runs for the various algorithms and the various instances. The higher the better. I put in bold the best values for each instance, and in italics the second best.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Exp</th>
<th>E-GeNePi</th>
<th>E-GrGA</th>
<th>E-GRASP</th>
<th>E-NSGA</th>
<th>E-PLS</th>
<th>E-HC</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>14</td>
<td>1.5</td>
<td>1.01</td>
<td>1.14</td>
<td>1.13</td>
<td>1.0</td>
<td>1.49</td>
</tr>
<tr>
<td>a_1_2</td>
<td>e14</td>
<td>12.66</td>
<td>11.55</td>
<td>9.9</td>
<td>10.61</td>
<td>6.21</td>
<td>8.74</td>
</tr>
<tr>
<td>a_1_3</td>
<td>e14</td>
<td>8.14</td>
<td>7.23</td>
<td>6.31</td>
<td>7.27</td>
<td>1.05</td>
<td>5.96</td>
</tr>
<tr>
<td>a_1_4</td>
<td>15</td>
<td>42.39</td>
<td>4.13</td>
<td>2.41</td>
<td>4.22</td>
<td>1.89</td>
<td>3.35</td>
</tr>
<tr>
<td>a_1_5</td>
<td>15</td>
<td>20.45</td>
<td>18.35</td>
<td>17.62</td>
<td>20.30</td>
<td>9.99</td>
<td>16.95</td>
</tr>
<tr>
<td>a_2_2</td>
<td>14</td>
<td>20.09</td>
<td>16.75</td>
<td>9.30</td>
<td>17.61</td>
<td>6.26</td>
<td>13.85</td>
</tr>
<tr>
<td>a_2_3</td>
<td>14</td>
<td>19.55</td>
<td>18.13</td>
<td>8.44</td>
<td>18.00</td>
<td>3.82</td>
<td>4.46</td>
</tr>
<tr>
<td>a_2_4</td>
<td>14</td>
<td>43.13</td>
<td>36.03</td>
<td>5.73</td>
<td>33.47</td>
<td>0.93</td>
<td>6.42</td>
</tr>
<tr>
<td>a_2_5</td>
<td>13</td>
<td>4.49</td>
<td>4.39</td>
<td>1.78</td>
<td>3.97</td>
<td>0.22</td>
<td>1.14</td>
</tr>
<tr>
<td>b_1</td>
<td>14</td>
<td>10.96</td>
<td>10.60</td>
<td>10.26</td>
<td>3.72</td>
<td>3.72</td>
<td>10.09</td>
</tr>
<tr>
<td>b_2</td>
<td>14</td>
<td>12.32</td>
<td>11.93</td>
<td>11.99</td>
<td>5.62</td>
<td>5.63</td>
<td>7.57</td>
</tr>
<tr>
<td>b_3</td>
<td>13</td>
<td>5.54</td>
<td>5.37</td>
<td>5.31</td>
<td>1.51</td>
<td>1.51</td>
<td>1.71</td>
</tr>
<tr>
<td>b_4</td>
<td>13</td>
<td>12.19</td>
<td>11.50</td>
<td>10.84</td>
<td>4.27</td>
<td>4.27</td>
<td>4.29</td>
</tr>
<tr>
<td>b_5</td>
<td>20</td>
<td>6.68</td>
<td>6.57</td>
<td>6.51</td>
<td>4.18</td>
<td>4.18</td>
<td>4.18</td>
</tr>
</tbody>
</table>

Table 5.2 shows that E-GeNePi finds the highest number of solutions (10 out of 14 instances) or the second highest (4 remaining instances): on average,
E-GeNePi finds 14.84% more solutions than the second best or best algorithms. We also notice a large improvement from the two-step method (GrGA) with an increase of 229.91% in the number of found solutions (and 41.96% without including a\_1\_1). GrGA, GRASP and NSGA have fluctuating results depending on the instance, and PLS does not perform well. The new thing we learn is that HC gets a fair number of solutions for easy and medium instances (they take less time to evaluate), but gets worse results for others.

### 5.5.3 Profiling

I plot in Figure 5.4 the evolution of the hypervolume over time for the different algorithms on the different instances. Each dot corresponds to a solution (or group of solutions) found.
5.5 Evaluation

Figure 5.4 – Continued from previous page

Figure 5.4 – Continued on next page
The first thing to notice is that algorithms based on GRASP (E-GeNePi, E-GrGA and E-GRASP) are aggressive at the start, due to the greedy nature of GRASP and the usage of weights that lead to a large spread over the research space. After 10-20% of the time allocated, E-GRASP though plateaus. We also see that E-GeNePi and E-GrGA have almost the same behaviour until a stage when NSGA-II reaches an elite population (no more improvements) while E-GeNePi increases the number of solutions thanks to its last step, PLS. E-GRASP mixes randomness and aims to get the best possible improvement, which leads it occasionally to have a good improvement (e.g., between 862s and 1,286s in a_1_2) when it finds randomly a good solution; but it can also end up stuck for a long time. E-NSGA shows it can only improve the hyper-
volume on small and medium instances, and this improvement is spoilt by the first phase, i.e., the generation of a population from the initial assignment which looks bad. This issue is less dramatic when the search space is small (i.e., for the small instances). Then the random generation does not seem too bad – as the hypervolume increases well and solutions are found. Because they require a long evaluation of the neighbourhood of a solution, algorithms based on a local search (E-PLS and E-HC) perform poorly. E-HC is relatively better though, especially for the smaller instances.

5.6 Conclusion

This chapter gives the first full formal definition of the multi-objective VM reassignment problem for large decentralised data centres. This problem is deemed important for many large organisations, where hosting departments have a certain degree of autonomy and the capital allocators’ preferences are not always aligned with the main objectives of the organisation. This problem is challenging given the size of real world IT infrastructures, the variety of individual capital allocators’ preferences and the complexity of the multi-objective resolution.

I have proposed E-GeNePi, a solution composed of two modules: one for the capital allocators of the various hosting departments (VCs) which aims at finding a placement of VMs on the VC’s PMs and improving this placement as much as they can (using a First Fit Decreasing algorithm and a Hill Climbing algorithm); and one for the decision makers at the top of the organisation, which aims at finding a broad set of good reassignments (i.e., Pareto, with no other solution that beats them on every objective). A comparison on a realistic dataset showed that E-GeNePi outperforms other systems implementing multi-objective algorithms in terms of quantity of solutions (+14.84% on average) and quality of the solutions set (+6.03% on average).

In the next chapter, I add the critical and challenging public cloud element to decentralised DCs, allowing them to decommission some of their workloads to the Cloud while taking into account the price fluctuation of cloud services.
In this chapter, I formally define the multi-objective VM reassignment problem for hybrid and decentralised data centres. I also propose H2-D2, a solution that uses the same multi-layer architecture as in Chapter 5 and an adaptation of GeNePi (the three-step metaheuristic described in Chapter 4) to suggest reassignment solutions to the various hosting departments and available public clouds.

6.1 Introduction

There has been a proliferation of cloud services in the past years, from virtual machines of different flavours to out-of-the-box platforms (e.g., ready to use Machine Learning tools). The many benefits of these cloud solutions [13], including but not limited to their cost, have accelerated the adoption of the Cloud for all sorts of companies [14]. However, modern large and often global organisations tend to ponder over outsourcing to the Cloud more than small and medium companies [14], with only 17% of them reported having 1000+ VMs in the Cloud. Some of the many reasons for this slow process are the complexity of their software systems [15], the types of their workloads [169]
6.1 Introduction

and the privacy/security of their data and products [219] – as well as the distribution and segmentation of the data centres of these large and global companies [5]: they possess many hosting departments with ‘competing’ or even ‘conflicting’ demands and requirements.

However, the hybrid cloud solution [16], i.e., mixing private infrastructure and public cloud services, is now seen as a potential way for these large companies [14] as it gives them the benefits of both worlds. On one hand the Cloud provides quick infrastructure provisioning and deployment [17], while on the other hand companies can still maintain their own infrastructure when exact characteristics of servers [18], performances [220] and reliability [169] are important.

In this chapter, I address the problem of multi-objective VM reassignment for large and hybrid decentralised data centres. I consider that data centres are decentralised, i.e., capital allocators of hosting departments express their own preferences and I also consider hosting some of the VMs in the Cloud as an option. Figure 6.1 shows an example of such hybrid decentralised data centres, with a private data centre composed of three VCs and a public cloud composed of two public cloud providers with several Cloud locations each.

I modify the two-level system E-GeNePi that is presented in Chapter 5 to propose a multi-objective VM reassignment system for large and hybrid de-
centralised DCs (H2-D2 stands for Hybrid algorithm for Hybrid Decentralised Data centres) which adds the critical and challenging public cloud element. This new element makes the problem more complex and H2-D2 has to take into account: (i) individual CAs’ preference, (ii) price fluctuation of cloud services and (iii) optimisation of the global infrastructure (at the general managers’ level).

In this chapter, I start by formally defining the multi-objective VM reassignment problem for hybrid and decentralised data centres (Section 6.2). Then, I describe my solution H2-D2 (Section 6.3). Next, I compare H2-D2 against various reassignment algorithms on a realistic dataset – inspired from the Google ROADEF/EURO Challenge 2012 and modified to make it more realistic for the complex context I address here (Sections 6.4 and 6.5). Finally, I make some concluding remarks (Section 6.6).

6.2 Formal Problem Definition

This section provides the first formalisation of the multi-objective VM reassignment problem in large and hybrid decentralised data centres. Such data centres are composed of: virtual machines, physical machines, virtual data centres, public cloud locations (Ps) with the type of VMs they offer (Fs). Each data centre is also defined by a large number of constraints (see below Section 6.2.1) and a set of objective functions (i.e., dimensions that define the search space). The optimisation problem itself is defined by two types of mappings of VMs on PMs: (i) more ‘general’ reassignment of VMs to either one of the VCs that compose the data centre or the public cloud, and (ii) ‘exact’ placement of VMs on the PMs belonging to the VCs where the VMs have been assigned.

A decentralised data centre is composed of a set $C$ of VCs: $C = \{c_1, \ldots, c_n\}$. Each $c_i$ is given a set $M_i \subseteq M$ of PMs, $M_i = \{m_1, \ldots, m_n\}$. Each $m_j \in M_i$ has several resources $r \in R$ (e.g., RAM, CPU, disk), in a limited capacity $Q_{m_j,r}$. Resources are either transient ($r \in TR \subseteq R$, such as RAM and disk) if they are consumed by both origin and destination PMs during a VM migration, or non-transient otherwise (e.g., CPU): $r \in R \setminus TR$. 

159
A hybrid data centre can outsource VMs to the different public cloud locations \( P_i \in \mathcal{P} \). I consider that there is no limit on the resources offered by public cloud locations, but each \( P_i \) varies on the type of flavours \( F_i \in \mathcal{F} \) of VMs they offer. Every flavour \( f \in F_i \) provides a quantity \( Q_{f,r} \) of resource \( r \in \mathcal{R} \) and has a price varying within the interval \( p_f = [p_{f,\underline{r}}, p_{f,\overline{r}}] \). To some extent, a flavour is a type of VM.

A hybrid decentralised data centre hosts a set \( \mathcal{V} \) of VMs, \( \mathcal{V} = \{v_1, \ldots, v_l\} \). Each VM \( v_k \) has to be reassigned to either a VC \( \text{Re}(v_k) = c_i | c_i \in \mathcal{C} \) in a PM \( M(v_k) = m_j | m_j \in \mathcal{M}_i \), or to a public cloud location \( \text{Re}(v_k) = P_l \) with a flavour \( M(v_k) = f_l \) (I also use the notation \( M_0(v_k) = m_j \) for the original PM of \( v_k \) in case of a migration). The quantity of resource \( r \) that every \( v_k \) needs is fixed to \( d_{k,r} \). VMs are sometimes grouped into services \( \mathcal{S} = \{s_1, \ldots, s_p\} \), with \( s_p = \{v_{1,p}, \ldots, v_{q,p}\} \). A service can be seen as a duplication of the same process to ensure the resiliency of the application to defects.

### 6.2.1 Constraints of the Problem

The problem of VM reassignment considers a large number of hard constraints at three different levels:

#### 6.2.1.1 VC Level

Every placement of VM on a PM is subject to a set of constraints. I have described those constraints in Chapter 5 and I just summarise them here:

- **Capacity**: requires that VMs do not exceed the resource capacity of the PM they are hosted on or they are migrating to/from (in case of transient resources).
- **Conflict**: reflects the fact that services running on several VMs may prefer to use a different PM for each VM.
- **Dependency**: when a service depends on another one, every VM of this first service needs to be in the neighbourhood of at least one VM from the second service.
6.2 Formal Problem Definition

6.2.1.2 Public Cloud Level

Deploying a VM to the Cloud is usually done through tools or APIs depending on the cloud providers, with service level agreements decided ahead of the deployment. Thus, there is little control over the characteristics of the hosts on which the VMs will be deployed (e.g., exact characteristics of the PMs, location). Although cloud providers are known for the quality and reliability of their infrastructures, it is difficult with the Cloud to address conflict and dependency constraints, making the Cloud irrelevant for certain VMs.

**Fitting VM flavour**: Public clouds list a certain number of VM flavours to choose from, with a pre-set quantity of resources that the user/client should not exceed. This number can be large as there might be VMs set for specific targets (CPU intensive, RAM intensive, GPU, etc.), but VM flavours often do not exactly match the needs of VMs. Therefore, a VM flavour with higher specifications has to be booked, leading to potential resource wastage (hence the $\leq$ sign in equation 6.1).

$$
\forall v \in V, \quad M(v) = f_i \quad \implies \quad \forall r \in R, \quad d_{v,r} \leq Q_{f_i r} \quad (6.1)
$$

**Conflicting VMs**: In private DCs, VMs that belong to the same service are considered conflicting, and cannot share the same host. On the other hand, public clouds behave as black boxes and do not provide any knowledge regarding the placement of VMs within their location. Therefore, it is not possible to send two VMs belonging to the same service to the same location of the public cloud without taking a risk of having them sharing the same host. To avoid this, I have to reassign VMs of the same service to different locations. \( \forall v_i, v_j \in V, \forall s_i \in S, \forall P_i, P_k \in P, (v_i, v_j) \in s_i^2, i \neq j: \)

$$
Re(v_i) = P_i, \quad Re(v_j) = P_k \quad \implies \quad l \neq k \quad (6.2)
$$

**Dependent VMs**: A service \( s_l \) depending on a service \( s_u \) (\( s_l \rightarrow s_u \)), obliges VMs of \( s_l \) to be in the neighbourhood of at least one VM from the service \( s_u \). Since there is little or no notion of neighbourhood (equipment sharing a
6.2 Formal Problem Definition

When it comes to public clouds, it is not possible to outsource any VM from the service $s_t$.

$$s_t \rightarrow s_{it} \implies \forall v \in s_t, \; R(e(v)) \notin \mathcal{P} \quad (6.3)$$

6.2.1.3 Manager Level

Most of the constraints concern the two levels above (i.e., VC and public cloud), with the exception of the spread constraints. Spread constraints express the fact that for security reasons, some services might require their VMs to be assigned to a number of VCs and/or public cloud locations higher than a value $\sigma_s$ that is considered safe (e.g., for redundancy reasons).

$$\forall s \in \mathcal{S}, \; \sum_{l \in \mathcal{C} \cup \mathcal{P}} \min \{1, |\{v \mid v \in s \land R(e(v)) = l\}|\} \geq \sigma_s \quad (6.4)$$

6.2.2 Objectives to Optimise

I now introduce the cost functions which define the various objectives. I consider in this work four objectives: two that depend on the characteristics of the VCs (i.e., reliability and electricity costs) and that are the same as those defined in Chapter 5, one that is given by the price of hosting VMs on the Cloud (i.e., cloud cost), and another one that is obtained from both VCs and public clouds’ characteristics (i.e., migration cost). In this chapter, I only define the novel cost that is the ‘Cloud Cost’ and I refer the reader to the previous chapter for the formal definition of the other objectives.

Note that when a VM $v$ is reassigned to a public cloud location $R(e(v)) = P_l$, using a VM flavour $M(v) = f_l$, it is only possible to give an estimation for the PM to PM migration cost depending on the public cloud location and the bandwidth associated with the VM flavour. This is also the case for the deployment time of a VM in the Cloud. Given that specifications of the hosting PMs are often unknown, I can only approximate this time based on the size of the VM and the processing power conferred by the VM flavour.
6.3 H2-D2

6.2.2.1 Cloud Cost

The (public) cloud cost sums up the price of every VM assigned to the public cloud. Since the fee of a VM flavour in a public cloud location is given as an interval, this objective is computed using interval arithmetic [221]:

\[
CC = \sum_{v \in V} \bigvee_{Re(v) \in P} \left[ \frac{p_{M(v)}}{M(v)} \right]
\]

(6.5)

Note that electricity and cloud costs cover different things in the finances of companies and are often impossible to compare. While they are both paid in "dollar", I see them as two independent objectives - which is really what they are in the enterprise world.

**Definition 4 (Reassignment)** A VM reassignment in a hybrid decentralised DC ReAssign is a mapping: ReAssign : \( V \rightarrow \mathcal{P} \cup \mathcal{C} \), such that ReAssign(v_k) \( \rightarrow c_i \) or ReAssign(v_k) \( \rightarrow P_l \), that ensures the satisfaction of the constraints defined in Section 6.2.1.

Given that I am in the context of a pure multi-objective optimisation, I aim at minimising all the objectives independently (i.e., minimising the multiple objective functions instead of minimising them in a combined fashion (i.e., as a single function). I obtain from the optimisation a set of non-dominated solutions (solutions that are better than the others on at least one particular objective), thus defining a Pareto front.

6.3 H2-D2

I present a system that addresses the multi-objective VM reassignment problem in the context of large and hybrid decentralised data centres. H2-D2 aims at satisfying the preferences of CAs of individual VCs while offering managers of the private DC the ability to extend their resources using public clouds, hence improving their VM placement solutions in the multi-objective search
6.3 H2-D2

Figure 6.2: Overview of H2-D2. The grey boxes represent the two main components: (i) the reassignment of VMs to VCs and cloud platforms and (ii) the exact placement of VMs in the servers of the VCs. Note that each solution is evaluated by cloud providers and VCs’ CAs as the companies’ managers may not know the exact price of hosting the VMs in the Cloud or the characteristics of each placement (depend on VCs’ CAs preferences).

space. H2-D2 has two different modules responsible for the reassignment of VMs and their placement (see grey boxes in Figure 6.2).

In this chapter, I mainly focus on the reassignment technique that is run at the manager level (i.e., decision makers). The placement algorithm that is run at the VC level in my case is based on: (i) a First Fit Decreasing algorithm for finding a feasible initial placement and (ii) a Hill Climbing algorithm for optimising the placement according to preferences of the CAs. I refer to the Chapter 5 for more details regarding this part. Notice that many algorithms for finding efficient VM placements exist in the literature [222] and can be used interchangeably. However, it is preferable to use one that replies quickly regarding the existence of a feasible placement, so that a reassignment can be quickly discarded.
6.3.1 Reassignment

Decision makers at the top-level of the private DC run an extension of GeNePi (my hybrid metaheuristic defined in Chapter 4) to handle four objectives of possibly interval values. Decision makers ensure that all the outsourced VMs satisfy the hard constraints. They also send all of the VMs reassigned within the private infrastructure to their corresponding VCs, where a placement is tried: if every VC succeeds in finding a valid placement, then the real cost of the solution is updated (the solution is possibly discarded if considered not good enough with regards to the four objectives). In the event that the reassignment is violating some constraints, GeNePi modifies it and tries resubmitting it.

GeNePi starts with GRASP’s constructive phase [115] as a first step: VMs are arranged decreasingly following their dependencies and requirements. The top ones are successively assigned to VCs that allow the satisfaction of the spread and dependency constraints and have a utility value beyond a threshold $a$. For every solution, a certain ratio of VMs that are not concerned by dependencies and that are eligible to exit the private DC are sent to non-conflicting public cloud locations. The higher the ratio, the more VMs are outsourced. This ratio is updated in a dynamic (using dichotomy to explore a large number of outsourcing degrees) and reactive (when constraints are violated, the algorithm considers that the limit of outsourcing capacity is reached, thus the ratio can only decrease) fashion.

Then, GeNePi applies NSGA [223]: a genetic algorithm which in addition to coping with interval objectives distinguishes itself by (i) selecting and making a tournament between four parents before mixing the winners (using different operators: crossover and mutation); (ii) mixing the offspring population (i.e., population resulting from an evolution) with the original one, thus only keeping the elite solutions for a faster convergence towards the optimal Pareto front. In my algorithm, I use One-point and Two-point ‘cuts’ as crossovers where random cut positions are selected in the parents’ chromosomes and their respective parts exchanged. I also use a mutation which randomly reassigns a VM to either a public cloud location or a VC.

Last but not least, GeNePi applies PLS [218]; a local search which only looks for new solutions in the neighbourhood of non-dominated solutions that
are the most isolated (as they are the most likely to refine the non-dominated set). GeNePi uses 1-exchange (changing the assignment of one VM at a time) and swap (exchanging assignments between two VMs) as operators for generating the neighbourhood of each solution.

I use the same parameters for GeNePi as in Chapter 4. Moreover, a ratio of the global execution time (i.e., the time limit) is allocated to each step which allows a better control over the execution time and a reactivity towards the complexity of the instance and allocate a same execution time for each step as in E-GeNePi (i.e., see previous chapter) without performing an extensive tuning analysis that could allow achieving a better performance. Therefore, a third of the execution time is given to GRASP. In the event that the size of NSGA’s initial population is not reached, this time can be extended up to half of the time limit. As NSGA achieves large improvements, I allow it the longest execution time with half of the global execution time. PLS gets only one sixth of the global time as it has a vocation to refine the Pareto front by finding more non-dominated solutions, and not to improve its quality.

### 6.4 Experimental Setup

In this section, I evaluate the performance of H2-D2 and state-of-the-art multi-objective VM reassignment systems for large and hybrid decentralised DCs. I use in this evaluation the large dataset provided by Google to the ROADEF/EURO challenge 2012.

I have modified the dataset provided in the challenge Google ROADEF/EURO 2012 to make it more realistic to large decentralised IT companies where various hosting departments “collaborate” and “compete” with the central managers (i.e., similarly to Chapter 5). In the current chapter I also add information about public clouds (VM flavours, i.e., resource capacities and price intervals).

Table 6.1 summarises the characteristics of the hybrid decentralised DCs in my dataset (every instance corresponds to one DC). The dataset consists of three sets of instances: (i) instances a_1_* (considered easy) have between 2 and 4 resources, 4 to 100 PMs, 100 to 1,000 VMs, 79 to 981 services, 4 to 50 VCs and 15 to 80 VM flavours in 3 to 9 public cloud locations; (ii) instances a_2_*
(considered medium) have 12 resources, 50 or 100 PMs, 1,000 VMs, 129 to 180 services, 25 VCs and 30 to 100 VM flavours in 3 to 10 public cloud locations; and (iii) instances b_ (considered difficult) have 6 or 12 resources, 100 or 500 PMs, 5,000 to 20,000 VMs, 1,732 to 15,025 services, 10 to 50 VCs and 12 to 117 VM flavours in 4 to 9 public cloud locations.

Table 6.1: Characteristics of the different data centres (one instance equals one data centre) and the allowed execution time for each of them.

<table>
<thead>
<tr>
<th>Instance</th>
<th># Resources</th>
<th># PMs</th>
<th># VMs</th>
<th># Services</th>
<th># VCs</th>
<th># Cloud Locations</th>
<th># VM Flavours</th>
<th>Exec Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>2</td>
<td>4</td>
<td>100</td>
<td>79</td>
<td>4</td>
<td>8</td>
<td>64 (8\times8)</td>
<td>30</td>
</tr>
<tr>
<td>a_1_2</td>
<td>4</td>
<td>100</td>
<td>1,000</td>
<td>980</td>
<td>4</td>
<td>3</td>
<td>15 (5\times3)</td>
<td>1,200</td>
</tr>
<tr>
<td>a_1_3</td>
<td>3</td>
<td>100</td>
<td>1,000</td>
<td>216</td>
<td>25</td>
<td>9</td>
<td>63 (7\times9)</td>
<td>1,200</td>
</tr>
<tr>
<td>a_1_4</td>
<td>3</td>
<td>50</td>
<td>1,000</td>
<td>142</td>
<td>50</td>
<td>8</td>
<td>80 (10\times8)</td>
<td>1,200</td>
</tr>
<tr>
<td>a_1_5</td>
<td>4</td>
<td>12</td>
<td>1,000</td>
<td>981</td>
<td>4</td>
<td>3</td>
<td>36 (12\times3)</td>
<td>1,200</td>
</tr>
<tr>
<td>a_2_1</td>
<td>3</td>
<td>100</td>
<td>1,000</td>
<td>1,000</td>
<td>1</td>
<td>8</td>
<td>56 (7\times8)</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_2</td>
<td>12</td>
<td>100</td>
<td>1,000</td>
<td>170</td>
<td>25</td>
<td>3</td>
<td>30 (10\times3)</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_3</td>
<td>12</td>
<td>100</td>
<td>1,000</td>
<td>129</td>
<td>25</td>
<td>10</td>
<td>60 (6\times10)</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_4</td>
<td>12</td>
<td>50</td>
<td>1,000</td>
<td>180</td>
<td>25</td>
<td>8</td>
<td>56 (7\times8)</td>
<td>3,600</td>
</tr>
<tr>
<td>a_2_5</td>
<td>12</td>
<td>50</td>
<td>1,000</td>
<td>153</td>
<td>25</td>
<td>10</td>
<td>100 (10\times10)</td>
<td>3,600</td>
</tr>
<tr>
<td>b_1</td>
<td>12</td>
<td>100</td>
<td>5,000</td>
<td>2,512</td>
<td>10</td>
<td>9</td>
<td>117 (13\times9)</td>
<td>7,200</td>
</tr>
<tr>
<td>b_2</td>
<td>12</td>
<td>100</td>
<td>5,000</td>
<td>2,462</td>
<td>10</td>
<td>3</td>
<td>24 (8\times3)</td>
<td>7,200</td>
</tr>
<tr>
<td>b_3</td>
<td>6</td>
<td>100</td>
<td>20,000</td>
<td>15,025</td>
<td>10</td>
<td>8</td>
<td>56 (7\times8)</td>
<td>7,200</td>
</tr>
<tr>
<td>b_4</td>
<td>6</td>
<td>500</td>
<td>20,000</td>
<td>1,732</td>
<td>50</td>
<td>4</td>
<td>12 (3\times4)</td>
<td>7,200</td>
</tr>
</tbody>
</table>

Instance a_1_3 for example represents a DC that has 3 resources, 100 PMs, 1,000 VMs, 216 services, 25 VCs and 9 public cloud locations offering 63 VM flavours altogether (the total number of VM flavours = number of VM flavours per location \times number of public cloud locations). All the instances have a number of public cloud locations and a number of VM flavours generated randomly within the intervals [3, 10] and [3, 14] following offers for ‘general purpose’ VMs by major public clouds. Table 6.1 also shows the maximum allowed execution time for each optimisation algorithm. These values were discussed with industry experts from the domain and reflect their common knowledge about the decision making process in large companies. In particular, they represent the consolidation of workloads done on a regular basis, e.g.,
monthly or quarterly and the time that can be given to the tools supporting capital allocators’ decision process.

I also use the same two metrics as in the previous chapters: the quantity of non-dominated solutions found (how many non-dominated solutions are found) and the quality of the found solutions (how much of the search space do they cover expressed using the hypervolume).

In my model, the exact price of hosting VMs in the Cloud is not known in advance but is taken from an interval of values (which reflects the fact that price in the Cloud is not always determined in advance). The reassignment solutions found by the algorithms then depend on the exact price of hosting VMs in the Cloud, which is represented in Figure 6.3 (left figure) by the broken segments, i.e., the intervals of cloud costs. The segments with black dots represent the solutions on the Pareto front (the good ones) and the segments with white dots represent the other (not interesting) solutions. Now, as the cloud cost depends on the exact price of hosting the VMs in the Cloud, the evaluation of the solutions formed by the algorithms may vary. To tackle this problem, I have created these different scenarios:

- one, that I call optimistic, where the price of hosting VMs in the Cloud is always the lowest. This corresponds to the case when the capital allocators are ‘lucky’ and have to pay less than expected for the VMs.
- on the contrary, the pessimistic scenario gives the highest possible price in the interval to all VMs.
- average scenario gives the average price to all of the VMs hosted in the Cloud.

Figure 6.3 shows the impact of the optimistic and pessimistic scenarios on the results of an algorithm: (i) the hypervolume (grey area) is different (bigger in the optimistic scenario) and (ii) the number of solutions also varies in the example (the solution d being on the Pareto front in the pessimistic case but not in the optimistic case).
6.4 Experimental Setup

Figure 6.3: The solutions found by an algorithm (on the left) have the form of intervals as the price of VMs in the Cloud is not known in advance. The two figures on the right show the results under the optimistic and pessimistic scenarios with the solutions on the Pareto front (black dots) and the hypervolume (grey area).

6.4.1 Other Algorithms

There is no direct comparison of H2-D2 against other state-of-the-art solutions – as, to the best of my knowledge H2-D2 is the first solution to the multi-objective VM reassignment in hybrid Decentralised DCs. However, H2-D2 performs two main tasks, reassignment and placement, which can be evaluated. I compare the reassignment phase of H2-D2 against solutions found in the literature. More precisely I compare H2-D2 against nine other “systems” where the reassignment module is not implementing the three-phase technique, as in H2-D2, but another state-of-the-art algorithm:

- eBSA [164]; a cutting-edge algorithm for the placement of VMs in hybrid Enterprise data centres, with a geographically distributed infrastructure. eBSA is an extension of a former approach [224] which com-
6.4 Experimental Setup

bines a Biased Importance Sampling (BSA) and a Cross-Entropy method (CE [165]). eBSA is based on three major concepts: (i) generating solutions probabilistically, (ii) biasing the generation of solutions to favour the creation of feasible ones, and (iii) optimising the solutions using a CE importance sampling. Given that eBSA was designed with a mono-objective optimisation in mind, I had to adapt it in my implementation in order to deal with multiple objectives. Therefore, I use the Pareto ranking algorithm described and implemented in [225] for the ranking of individuals and the generation of importance probabilities during the importance sampling phase.

- different elements of my three-phase optimisation technique (GeNePi) taken individually (to evaluate the relative importance of each of them).

The first algorithm; GRASP is run with the same maximally spread weighted sum vectors as in GeNePi during the whole execution time. The second algorithm; NSGA is run with the same parameters as in GeNePi, with a third of the execution time being dedicated to randomly generating its initial population. The third algorithm; PLS is run for the entire execution time, but unlike in GeNePi, it is run on all the iterative Pareto front solutions. Given that GeNePi is a three-step method, I also decided to compare it against GrNSGA; a combination of its two first algorithms: GRASP (one-third of the time limit) followed by NSGA (two-thirds of the time limit).

- other common and classical multi-objective optimisation techniques. I start with a Strength Pareto Evolutionary Algorithm (SPEA2 [75]); a well-known genetic algorithm, that is extensively used to compare existing evolution-based methods. As is the case with NSGA, SPEA2 is used in two different ways: (i) SPEA with a randomly generated initial population and (ii) GrSPEA with an initial population generated using GRASP. In addition, I compare my three-step algorithm to the multi-objective Cross-Entropy (CE [225]) method. I finish with HC which runs iteratively a Hill Climbing algorithm with a weighted sum objective, taking a different vector of weights on a maximally spread basis at every iteration.

All the algorithms used here are developed in C++ and the experiments are run on a machine with 24 Intel® Xeon® 2.20GHz CPUs and 64GB of
RAM, running Ubuntu 12.4 LTS (64bit).

6.5 Evaluation

I compare in this section the Pareto fronts found by H2-D2 and the other systems presented above, both quantitatively (number of solutions) and qualitatively (space covered by the solutions, i.e., hypervolume). The average of 10 runs per algorithm and per instance is taken. I also profile how the different algorithms optimise as it is interesting to know their individual behaviour and their composition in a hybrid technique.

6.5.1 Quantitative and Qualitative Evaluation

I compare the different reassignment algorithms described earlier both quantitatively and qualitatively. I show the number of non-dominated solutions found by every algorithm in Table 6.2. Given that the exact price of every VM flavour (in every public cloud) is not known but is to be picked from an interval, it is not possible to compare solutions given by the different algorithms as the price fluctuation has an impact on what is the best placement. In this study, I consider two scenarios: (i) one scenario that I call optimistic, where the price of hosting VMs is set at the minimum value in the range and (ii) one scenario, pessimistic, where the price is set at the maximum. This gives an insight on the qualitative results of the algorithms in the extreme cases: when the cost of hosting VMs in the Cloud is low or high. Table 6.3 shows the results obtained by every algorithm under those two scenarios, in terms of hypervolume.

Figures 6.4 and 6.5 summaries the results and show the worst, best and average ranking for each of algorithms on the ROADEF instances respectively in terms of number of non-dominated solutions and hypervolume.

We notice that GeNePi outperforms all of the other algorithms in terms of hypervolume when considering the optimistic scenario, with an improvement of 19.72% on average over the second best algorithm. GeNePi also outperforms the other algorithms on most of the instances with the exception of \( b_4 \) in the pessimistic scenario and achieves an improvement in hypervolume of
6.5 Evaluation

Table 6.2: Average results of 10 runs in terms of number of non-dominated solutions.

<table>
<thead>
<tr>
<th>Inst</th>
<th>Init</th>
<th>GeNePi</th>
<th>GRASP</th>
<th>NSGA</th>
<th>SPEA</th>
<th>GrNSGA</th>
<th>GrSPEA</th>
<th>PLS</th>
<th>HC</th>
<th>CE</th>
<th>eBSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1_1</td>
<td>1</td>
<td>951</td>
<td>659</td>
<td>462</td>
<td>405</td>
<td>525</td>
<td>497</td>
<td>739</td>
<td>573</td>
<td>127</td>
<td>128</td>
</tr>
<tr>
<td>a_1_2</td>
<td>1</td>
<td>336</td>
<td>238</td>
<td>6</td>
<td>18</td>
<td>192</td>
<td>127</td>
<td>33</td>
<td>8</td>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>a_1_3</td>
<td>1</td>
<td>89</td>
<td>39</td>
<td>31</td>
<td>37</td>
<td>68</td>
<td>57</td>
<td>30</td>
<td>30</td>
<td>19</td>
<td>40</td>
</tr>
<tr>
<td>a_1_4</td>
<td>1</td>
<td>4,877</td>
<td>574</td>
<td>1,536</td>
<td>1,674</td>
<td>1,855</td>
<td>1,372</td>
<td>3,744</td>
<td>125</td>
<td>18</td>
<td>48</td>
</tr>
<tr>
<td>a_1_5</td>
<td>1</td>
<td>206</td>
<td>226</td>
<td>3</td>
<td>10</td>
<td>125</td>
<td>87</td>
<td>9</td>
<td>77</td>
<td>7</td>
<td>42</td>
</tr>
<tr>
<td>a_2_1</td>
<td>1</td>
<td>555</td>
<td>284</td>
<td>3</td>
<td>43</td>
<td>206</td>
<td>168</td>
<td>12</td>
<td>15</td>
<td>8</td>
<td>33</td>
</tr>
<tr>
<td>a_2_2</td>
<td>1</td>
<td>395</td>
<td>225</td>
<td>10</td>
<td>25</td>
<td>483</td>
<td>331</td>
<td>29</td>
<td>193</td>
<td>11</td>
<td>84</td>
</tr>
<tr>
<td>a_2_3</td>
<td>1</td>
<td>103</td>
<td>33</td>
<td>47</td>
<td>53</td>
<td>50</td>
<td>48</td>
<td>36</td>
<td>26</td>
<td>19</td>
<td>51</td>
</tr>
<tr>
<td>a_2_4</td>
<td>1</td>
<td>183</td>
<td>93</td>
<td>110</td>
<td>124</td>
<td>142</td>
<td>116</td>
<td>149</td>
<td>21</td>
<td>18</td>
<td>49</td>
</tr>
<tr>
<td>a_2_5</td>
<td>1</td>
<td>135</td>
<td>92</td>
<td>61</td>
<td>74</td>
<td>117</td>
<td>101</td>
<td>44</td>
<td>55</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>b_1</td>
<td>1</td>
<td>36</td>
<td>29</td>
<td>9</td>
<td>11</td>
<td>31</td>
<td>29</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>b_2</td>
<td>1</td>
<td>57</td>
<td>32</td>
<td>9</td>
<td>15</td>
<td>36</td>
<td>28</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>30</td>
</tr>
<tr>
<td>b_3</td>
<td>1</td>
<td>22</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>17</td>
<td>16</td>
<td>18</td>
<td>7</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>b_4</td>
<td>1</td>
<td>21</td>
<td>16</td>
<td>11</td>
<td>15</td>
<td>15</td>
<td>12</td>
<td>23</td>
<td>9</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

18.22% on average against the second best algorithm. Although the number of non-dominated solutions is a less important metric than the hypervolume as an algorithm can find more solutions with a poor quality, GeNePi finds more non-dominated solutions than the other algorithms in 11 cases out of 14 and GeNePi consistently attains at least the second best results. This corresponds to an improvement of 29.99% on average in comparison to the second best results.

We also see that both GrNSGA and GrSPEA get relatively good improvements in terms of hypervolume (close to results of GeNePi) on all of the instances, but GrNSGA and GrSPEA do not obtain as many non-dominated solutions as GeNePi (GeNePi gets respectively 61.43% and 100.02% more non-dominated solutions than GrNSGA and GrSPEA on average). GrNSGA is better in general than GrSPEA as it outperforms it in hypervolume on all of the instances in the pessimistic scenario (GrNSGA gets an improvement of 21.62% in hypervolume on average) and on 11 instances out of 14 in the optimistic scenario (GrNSGA gets an improvement of 4.11% in hypervolume on average). However, the difference in terms of the number of non-nominated
GRASP gets a fair number of non-dominated solutions, but with an average quality (GRASP reaches 54.88% and 65.25% of the hypervolume obtained
6.5 Evaluation

Figure 6.4: Best, average and worst ranking for each algorithm on the ROADEF instances in terms of number of non-dominated solutions.

NSGA and SPEA do not perform well, both quantitatively and qualitatively. NSGA and SPEA only reach respectively 29.38% and 32.41% in the pessimistic scenario, and 33.72% and 34.21% in the optimistic scenario of the hypervolumes obtained by GeNePi on average. NSGA and SPEA do not perform well in terms of number of non-dominated solution either as GeNePi gets respectively $\sim 26$ and $\sim 5$ times more non-dominated solutions than NSGA and SPEA on average. This is mainly due to the bad quality of the random generation of the initial population that affects negatively genetic algorithms. This effect can be seen when comparing their results with those obtained using a better initial population generated by applying GRASP. GrNSGA achieves on the optimistic and pessimistic scenarios respectively $\sim 5$ and $\sim 27$ times the improvement of NSGA in terms of hypervolume on average. GrNSGA also gets $\sim 14$ times more non-dominated solutions in comparison to NSGA on average, whereas GrSPEA achieves on the optimistic and pessimistic scenarios respectively $\sim 8$ and $\sim 10$ times the improvement of SPEA in terms of
Figure 6.5: Best, average and worst ranking for each algorithm on ROADEF instances in terms of obtained hypervolume.

hypervolume on average. GrSPEA also gets $\sim$2 times more non-dominated solutions in comparison to SPEA. Unlike GrNSGA and GrSPEA, SPEA performs relatively better than NSGA in most of the cases when given randomly generated initial populations, as SPEA gets 6.97% better hypervolume in the optimistic scenario and 153.91% more non-dominated solutions on average.

PLS on its part gets a large number of non-dominated solutions and even has the second best number of non-dominated solutions in 5 cases. However, the quality of these solutions is varying from one instance to another. PLS achieves a bad hypervolume on small instances, however, this trend changes when it comes to larger ones, even outperforming GeNePi and the other algorithms on the pessimistic scenario of instance $b_4$.

eBSA does not perform as well as expected as it only reaches 33.92% and 33.97% of GeNePi’s hypervolume on average respectively on the optimistic and pessimistic scenarios. eBSA gets an average number of non-dominated solutions (always more than 10), but their number is very small when compared to GeNePi (GeNePi gets $\sim$11 times more non-dominated solution on
average). Despite eBSA’s average performance, it is still outperforming basic genetic algorithms as eBSA gets 28.12% and 15.64% better hypervolume in the optimistic scenario, while getting 259.22% and 37.43% more non-dominated solutions when compared respectively with NSGA and SPEA. We clearly notice the advantage of biasing the importance sampling when looking at the performance of CE. CE achieves a poor performance both in terms of hypervolume and number of non-dominated solutions.

Jointly with CE, HC performs the worst overall, both quantitatively and qualitatively, despite some improvements in results when applied on some instances (e.g., a_1_1 and a_1_3).

We also notice large differences between hypervolumes of the optimistic and the pessimistic scenarios in same instances relatively to the improvements that are made by the different algorithms. These differences are reaching up to 63.77% in case of NSGA on average, which shows the impact of cloud price fluctuations and the importance of considering the cloud price as an interval rather than a fixed cost. NSGA is followed by CE and HC with differences of 50.29% and 50.08% respectively on average. GrNSGA, eBSA and GeNePi are the best in terms of difference of hypervolumes between the optimistic and pessimistic scenarios with 11.30%, 12.63% and 15.45% respectively thanks to the large number of non-dominated solutions they find on every instance, which tone down the effect of VM prices’ volatility in the Cloud.

### 6.5.2 Profiling

In this section I am interested in profiling the improvement in terms of hypervolume brought by the different reassignment algorithms on the different instances. I am particularly interested in the shape of the improvement curves. I plot in Figure 6.6 the evolution of the hypervolume improvement over time made by each algorithm on the studied instances. Given the complexity of showing the evolution of hypervolume as an interval, I choose to only plot the hypervolume when the price of every VM flavour in the public cloud is at its average. Each dot corresponds to a solution (or a set of solutions) found by the algorithm.
6.5 Evaluation

Figure 6.6 – Continued on next page
6.5 Evaluation

Figure 6.6 – Continued from previous page

Figure 6.6 – Continued on next page
6.5 Evaluation

Figure 6.6 – Continued from previous page

b_3

Figure 6.6: Evolution of the average hypervolume of 10 runs obtained using the different algorithms on the different instances over time when setting the VM prices to their average. Each dot corresponds to a solution (or a set of solutions) found by the algorithm.

As what has been seen in Table 6.3, GeNePi outperforms the other algorithms at the end of the time limit on all of the instances (with the exception of b_4) even when considering the average VM pricing scenario. GeNePi is followed closely by GrNSGA and GrSPEA.

Also, algorithms based on GRASP (i.e., GRASP, GrNSGA, GrSPEA and GeNePi) give better results due to their greedy nature and the usage of maximally spread weight vectors in their optimisation. Thus, improving significantly the hypervolume during the first fifth of the execution time. However, they often plateau until starting the next optimisation phase (i.e. either NSGA or SPEA) or continue on the same trend until the end of the execution time in the case of GRASP.

The NSGA step in both GrNSGA and GeNePi brings a quick and important improvement in hypervolume until reaching an elite population where only a marginal improvement can be obtained using the crossover and mutation operators. GrSPEA also improves the hypervolume significantly, however, not to the same extent as is observed for the GrNSGA’s. This is mainly due to the ranking procedure in SPEA which favours mostly the elite and the most spread individuals rather than the introduction of new ones and therefore leads to a premature convergence. Furthermore, we notice that GrNSGA
takes longer than GrNSGA to make any improvement between the different generations, which is mostly due to the time complexity of SPEA’s ranking procedure. This ‘time-wastage’ is especially noticeable at the start-up of the SPEA part as the initial population received from GRASP can be of a larger size than the offspring populations, thus taking longer to rank its individuals.

GeNePi copes with the convergence of the population in GrNSGA and the stagnation of the hypervolume by introducing a PLS step that increases the number of non-dominated solutions and gains some extra hypervolume improvement.

NSGA struggles to improve the hypervolume during the first third of its execution time. This corresponds to the random part that does not find solutions or finds some that are not interesting enough. After the random phase, we often notice an improvement in the hypervolume thanks to the genetic algorithm step. However, NSGA never reaches results obtained with GrNSGA due to the handicap of the random generation of the initial population.

As with NSGA, SPEA also struggles to improve the hypervolume during the first third of its execution time and does not reach GrSPEA’s results. In addition, SPEA takes longer to improve its initial population. However, despite being given a poor initial population, SPEA’s ranking procedure allows it to achieve a better improvement than NSGA.

CE’s and eBSA’s use of a random generation for their initial individuals does not bring much improvement either. After that phase, we see that CE is not improving significantly as it mostly finds non-feasible solutions. However, thanks to biasing the importance sampling, eBSA generates more feasible individuals. These individuals allow eBSA to achieve significant improvements that even outperform those of NSGA and SPEA in most cases. However, without ever reaching GrNSGA’s and GrSPEA’s results though. In terms of time complexity, we see that both CE and eBSA take some time (longer than the ranking in SPEA) to start generating individuals, which indicates that the Cross-Entropy method is very expensive in time. We also see that eBSA usually takes a bit longer than CE as biasing the importance sampling also adds a small but existing time complexity.

PLS goes through several improvements which are oftentimes marginal, particularly on small and medium instances. However, we see an increase in
6.6 Conclusion

performance on very large scale instances (i.e., \( b_3 \) and \( b_4 \)). HC shows the opposite behaviour to that of PLS. HC does not make many improvements, but makes more noteworthy ones when it makes any.

6.6 Conclusion

This chapter is the first attempt at addressing the complex options that capital allocators and managers of the infrastructure of large IT companies now face: (i) they can either deploy some of their workloads in the Cloud or keep them locally; (ii) the cost of hosting VMs in the Cloud may vary over time which makes a comparison of solutions difficult; (iii) internally these large, often geographically distributed companies are composed of multiple hosting departments with various local preferences; (iv) any reassignment of the VMs can be seen from different perspectives (objectives) that cannot be compared.

I have proposed H2-D2, a hybrid algorithm which reassigns VMs to the Cloud or internal hosting department using a succession of three optimisation algorithms (greedy, genetic algorithm and local search) and the placement of VMs in the hosting departments using a greedy routine followed by a Hill Climbing algorithm. I have extended a previous dataset proposed by Google to the ROADEF/EURO challenge and added information about public cloud services. The comparison against nine multi-objective placement algorithms shows that H2-D2 outperforms them both quantitatively (H2-D2 gets 29.99% more non-dominated solutions on average than the second best algorithm) and qualitatively (H2-D2 gets +19.72% and +18.22% improvement in hypervolume on average respectively in two extreme scenarios).

In the next chapter, I summarise the thesis, give some concluding remarks and provide some possible directions for future work.
This chapter concludes this thesis. First, I describe the contributions that emerged from this work. Then, I give some directions for possible future works.

### 7.1 Summary and Contributions of the Thesis

The first contribution of this thesis was to formally define the multi-objective VM reassignment in centralised data centres (as a multi-objective mixed-integer linear problem) by considering that top managers have full control of their geographically distant data centres. My definition is based on the single objective VM reassignment that was proposed by Google at the ROAD-EF/EURO challenge and includes several hard constraints: capacity (taking into account the effect of VM live migrations on transient resources), conflicts, dependencies and spread constraints. I also aimed at optimising three different objectives independently: electricity, reliability and migration costs.

The second contribution of this thesis was to study the applicability and the performance of a linear solver (CPLEX) and a state-of-the-art Constraint-Based hybrid algorithm (CBLNS) on the multi-objective VM reassignment
7.2 Possible Directions for Future Work

I aim in my VM reassignment problem to optimise three of the highly regarded objectives by practitioners and most commonly used in the literature. Although I based the definition of these objectives on widely used models that are proven to capture well what they represent, they are still simplistic
7.2 Possible Directions for Future Work

enough to be computed with simple functions. With the exception of the work on exact and hybrid solutions for the multi-objective VM reassignment in centralised data centres in Chapter 3, all the solutions that I propose throughout this thesis do not combine the objectives they optimise and they are totally agnostic of whether these ones are linear, non-linear or black-boxes. Therefore, I would like to explore more complex objectives by incorporating more parameters to the computation of the ones I already consider. For instance, the electricity consumption could be augmented by incorporating different elements such as the cooling of the data centre and electricity used for inter-VM network communications. The safety capacity that I use to measure the machine’s reliability to fulfil its tasks could also be modified to take into account a historical/predictive workload model, instead of a fixed safety value. In addition to the reliability cost that captures well the risk that a machine fails to execute its workload, it is not flexible for data centres that run VMs on behalf of clients. These data centres sign SLAs with some customers which make them more sensitive and more critical in case of failure. Therefore, VMs should not be considered in a similar way during the reassignment, but should instead prioritise and dedicate more attention to VMs bound by SLAs.

Modelling direct interactions between hosting departments is also something I would like to study and model in the near future. As it is adopted in large decentralised data centres, capital allocators tend sometimes to collaborate with their counterparts in other hosting departments. This collaboration often happens between different departments on a one-to-one basis, in the form of unstructured and informal discussions, without passing through an agreement at the management level. A department could decide to decommission some VMs to the benefit of another one if both see an incentive in doing so. For instance, some departments would want to reassign some VMs that require a lot of energy if their data centre is in a location with a high electricity price, and another department, with a data centre in a location with cheap electricity or with its own renewable energy sources, would see an incentive to host these VMs providing some monetary compensation.

The advantage of the multi-objective VM reassignment is that it provides decision makers with an overview of the optimisation capabilities in their infrastructure and allows them to plan their new reassignments according to their preferences. However, given the possibly large number of VMs that are
reassigned in certain solutions (a drastic change in configuration) and the absence of models that capture all the constraints of a large data centre, verifying the applicability of these reassignments and testing their performance before applying them becomes a necessity. Therefore, I would like to explore in the future the verification and testing of VM reassignments in large scale data centres before selecting them to be applied in production. In particular, I would like to apply model-based methods to generate the necessary test scenarios and use cases and apply them directly to the system, instead of performing them in a simulated environment with possibly less accuracy.

Last but not least, given the good performance that GeNePi achieves on the multi-objective VM reassignment problem and that GeNePi is a combination of three adaptable metaheuristics (i.e., GRASP, NSGA-II and PLS), I am interested in applying it to different problems from other contexts such as Search-Based Software Engineering (e.g., software testing, software product lines, etc.) where they often face large scale multi-objective problems.
REFERENCES


[34] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $\text{ffd}(i) \leq 11/9 \text{opt}(i) + 6/9$. In Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, pages 1–11. 2007. 14


[69] El-Ghazali Talbi. Metaheuristics: from design to implementation, volume 74. 2009. 21


[87] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm research: A history and analysis, 1998. 25

REFERENCES


195


[185] Qinghua Zheng, Rui Li, Xiuqi Li, Nazaraf Shah, Jianke Zhang, Feng Tian, Kuo-Ming Chao, and Jia Li. Virtual machine consolidated placement based on multi-objective biogeography-based optimization. FGCS, pages 95–122, 2016. 57

[186] Rui Li, Qinghua Zheng, Xiuqi Li, and Jie Wu. A novel multi-objective optimization scheme for rebalancing virtual machine placement. In CLOUD, 2016. 58


REFERENCES


[207] R Raju, J Amudhavel, Nevedha Kannan, and M Monisha. A bio inspired
energy-aware multi objective chiropteran algorithm (eamoca) for hybrid cloud computing environment. In ICGCCEE, pages 1–5, 2014. 68

[208] Chengyu Hu, Hong Yao, Chao Liu, and Deze Zeng. Executing time and cost-aware task scheduling in hybrid cloud using a modified de algorithm. In ISICA, pages 74–83, 2016. 68


[222] Xi Li, Anthony Ventresque, John Murphy, and James Thorburn. A fair comparison of VM placement heuristics and a more effective solution. In ISPDC, pages 35–42, 2014. 164

