Curso 2012/13
CIENCIAS Y TECNOLOGÍAS/4
I.S.B.N.: 978-84-15910-59-6

CARLOS SEGURA GONZÁLEZ

Técnicas de Optimización Paralelas. Esquema Híbrido basado en Hiperheurísticas y Computación Evolutiva

Directora
COROMOTO LEÓN HERNÁNDEZ

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

Acknowledgements

First of all, I would like to thank my advisor Coromoto León for her support and encouragement. Today, research is rarely carried out by one person alone. For this reason, much of the work has been done in collaboration with other researchers. I would like to thank everyone who has collaborated in this research. Specially I would like to thank Gara Miranda and Eduardo Segredo. Most of the research concerning hyperheuristics was done with Gara Miranda, while most of the research concerning multiobjectivisation was done with Eduardo Segredo.

This thesis is dedicated to my parents and brothers for their endless love, support and encouragement. Finally, I would like to make an special mention to my friends for their understanding when I have "disappeared" and for their encouragement.

I am also grateful to the Spanish Ministry of Education, Culture and Sport for grant FPU-AP2008-03213, which made this research possible. This work was also supported by the EC (FEDER) and the Spanish Ministry of Education, Culture and Sport as part of the 'Plan Nacional de I+D+i', with contract numbers: TIN2005-08818-C04-04, TIN2008-06491-C04-02, and TIN2011-25448. The Canary Government partially funded this work through the PI2007/015 research project. I also want to thank the HPC-EUROPA project (RII3-CT-2003-506079), and the HPC-EUROPA2 project (number: 228398) supported by the European Community Research Infrastructure Action. I would like to acknowledge the University of La Laguna for the usage of the SAII (Servicio de Apoyo Informático a la Investigación) computational systems.

Carlos Segura González

Abstract

Optimisation is the process of selecting the best element from a set of available alternatives. Solutions are termed good or bad depending on its performance for a set of objectives. Several algorithms to deal with such kind of problems have been defined in the literature. Metaheuristics are one of the most prominent techniques. They are a class of modern heuristics whose main goal is to combine heuristics in a problem independent way with the aim of improving their performance. Metaheuristics have reported high-quality solutions in several fields. One of the reasons of the good behaviour of metaheuristics is that they are defined in general terms. Therefore, metaheuristic algorithms can be adapted to fit the needs of most real-life optimisation. However, such an adaptation is a hard task, and it requires a high computational and user effort.

There are two main ways of reducing the effort associated to the usage of metaheuristics. First, the application of hyperheuristics and parameter setting strategies facilitates the process of tackling novel optimisation problems and instances. A hyperheuristic can be viewed as a heuristic that iteratively chooses between a set of given low-level metaheuristics in order to solve an optimisation problem. By using hyperheuristics, metaheuristic practitioners do not need to manually test a large number of metaheuristics and parameterisations for discovering the proper algorithms to use. Instead, they can define the set of configurations which must be tested, and the model tries to automatically detect the best-behaved ones, in order to grant more resources to them. Second, the usage of parallel environments might speedup the process of automatic testing, so high quality solutions might be achieved in less time.

This research focuses on the design of novel hyperheuristics and defines a set of models to allow their usage in parallel environments. Different hyperheuristics for controlling mono-objective and multi-objective multi-point optimisation strategies have been defined. Moreover, a set of novel multiobjectivisation techniques has been proposed. In addition, with the aim of facilitating the usage of multiobjectivisation, the performance of models that combine the usage of multiobjectivisation and hyperheuristics has been studied.

The proper performance of the proposed techniques has been validated with a set of well-known benchmark optimisation problems. In addition, several practical and complex optimisation problems have been addressed. Some of the analysed problems arise in the communication field. In addition, a packing problem proposed in a competition has been faced up. The proposals for such problems have not been limited to use the problem-independent schemes. Instead, new metaheuristics, operators and local search strategies have been defined. Such schemes have been integrated with the designed parallel hyperheuristics with the aim of accelerating the achievement of high quality solutions, and with the aim of facilitating their usage. In several complex optimisation problems, the current best-known solutions have been found with the methods defined in this dissertation.

Contents

List of Algorithms

List of Figures

List of Tables

List of Acronyms

2DPP	Two-Dimensional Packing Problem
ACO	Ant Colony Optimisation
ADI	Average Distance to Individuals
APP	Antenna Positioning Problem
CHC	Eshelman's cross generational elitist selection, heterogeneous recombination, and cataclysmic mutation
CGA	Cellular Genetic Algorithm
CMOS	Complementary Metal-Oxide-Semiconductor
COW	Clusters of Workstations
CPU	Central Processing Unit
CSP	Cutting Stock Problem
DBI	Distance to Best Individual
DC	Divide-and-Conquer
DCN	Distance to Closest Neighbour
DE	Differential Evolution
DFCN	Delayed Flooding with Cumulative Neighbourhood
EA	Evolutionary Algorithm
EAIPS	Evolutionary Algorithm with Increasing Population Size
EGA	Equilibrium Genetic Algorithm
EP	Evolutionary Programming
ES	Evolution Strategies
ESN	Evolution Strategy with NSGA-II
FAP	Frequency Assignment Problem
GA	Genetic Algorithm
GAA	Genetic Annealing Algorithm
GECCO	Genetic and Evolutionary Computation Conference
GEN-S	Generational with Elitism Selection
GP	Genetic Programming
GRASP	Greedy Randomised Adaptive Search Procedure
GSM	Global System for Mobile Communications

HPC	High Performance Computing
HUX	Half Uniform Crossover
IBEA	Indicator-Based Evolutionary Algorithm
ILS	Iterated Local Search
IX	Interference-based Crossover
LSHR	Local Search with Heuristic Restart
MANET	Mobile Ad-Hoc Networks
MA	Memetic Algorithm
METCO	Metaheuristic-based Extensible Tool for Cooperative Optimisation
MIFAP	Minimum Interference Frequency Assignment Problem
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MM	Mapping Mutation
MOCell	Multi-Objective Cellular Genetic Algorithm
MOEA	Multi-Objective Evolutionary Algorithm
MOFAP	Minimum Order Frequency Assignment Problem
MOGA	Multi-Objective Optimisation Genetic Algorithm
MOP	Multi-Objective Optimisation Problem
MOPSO	Multi-Objective Particle Swarm Optimisation
MSFAP	Minimum Span Frequency Assignment Problem
MPI	Message Passing Interface
NM	Neighbour-based Mutation
NOW	Networks of Workstations
NPGA	Niched Pareto Genetic Algorithm
NSGA	Non-Dominated Sorting Genetic Algorithm
NSGA-II	Non-Dominated Sorting Genetic Algorithm II
NSDEMO	Non-dominated Sorting Differential Evolution for Multiobjective Optimization
NUMA	Non-Uniform Memory Access
OpenMP	Open Multi-Processing
OPX	One Point Crossover
PAES	Pareto Archived Evolution Strategy
PBIL	Population-based Incremental Learning
PM	Polynomial Mutation
PMOEA	Parallel Multi-Objective Evolutionary Algorithm
PSO	Particle Swarm Optimisation
PVM	Parallel Virtual Machine
QoS	Quality of Service

RLD	Run-Length Distribution
RW-S	Replace Worst Selection
SA	Simulated Annealing
SBX	Simulated Binary Crossover
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SS	Scatter Search
SSX	Two-Dimensional Substring Crossover
SPEA	Strength Pareto Evolutionary Algorithm
SPEA2	Strength Pareto Evolutionary Algorithm 2
SPMD	Single-Program Multiple-Data
SSGA	Steady-State Genetic Algorithm
SS-S	Steady-State Selection
TS	Tabu Search
UM	Uniform Mutation
UX	Uniform Crossover
VEGA	Vector Evaluated Genetic Algorithm
VND	Variable Neighbourhood Descent
VNS	Variable Neighbourhood Search
2DPP	Two-Dimensional Packing Problem

Part I

Fundamentals and Background

Chapter
1

Introduction

This chapter introduces the main concepts used throughout the dissertation. The definition of optimisation problem as well as a brief introduction to some of the most extended optimisation strategies is offered. In addition, the set of metrics used to validate the proposals are summarised. The solution of most of the optimisation problems addressed in this research involves the usage of computationally expensive procedures. In order to reduce the time required to solve them, a parallel model has been proposed. Thus, an introduction to high performance computing and parallel computing is also presented. The main objectives and contributions achieved during the research are summarised in this chapter. Finally, the overview of the dissertation is described.

1.1 Optimisation Problems

Optimisation is a key topic in computer science, artificial intelligence, operational research and several other related fields [62]. In these fields, optimisation is the process of trying to find the best possible solution to solve a problem. Thus, it turns out that an optimisation problem is a problem for which there are different possible solutions, and there is a clear notion of solution quality. Solving an optimisation problem involves finding the best solution from all feasible solutions regarding certain objectives and subject to certain constraints. Computational optimisation is crucial in many fields of science and technology as well as in finance, business and medicine [154]. A solution to an optimisation problem is generated by taking several decisions that depend on the problem domain. For instance, the deployment of a wireless network might involve several decisions as selecting the places where the set of antennas must be placed, or assigning their frequencies; while in a scheduling

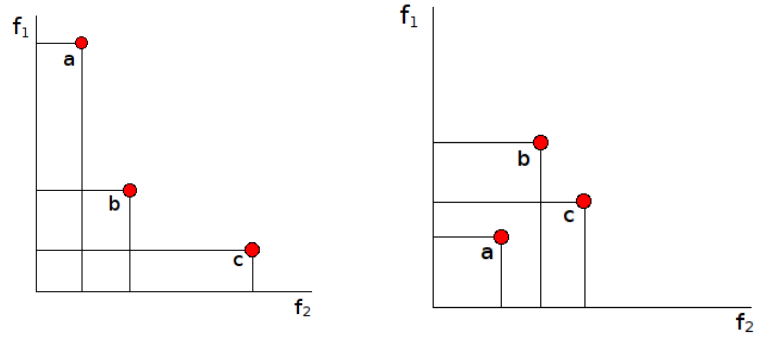
problem, the decisions might involve selecting the timetable for different lessons, or selecting the granted resources. Regardless of the given optimisation problem, such decisions are usually expressed as a set of decision variables. Therefore, a solution of the problem is generated by assigning specific values to such a set of variables. This set of variables is usually called the decision vector.
Given an optimisation problem P, the quality of a decision vector x is determined by a function f. When the aim of the optimisation problem is to maximise f, the optimisation function f is usually said to be a fitness function. In the cases where f must be minimised, f is usually said to be a cost function. However, some authors use the term fitness function regardless of the optimisation direction. Thus, in order to avoid misunderstanding, it is better to explicitly define the optimisation direction of f. Based on the function f, optimisation problems can be classified as mono-objective or multi-objective optimisation problems. In mono-objective optimisation problems f is a scalar function, while in multi-objective optimisation problems f is a vector function. Several formal definitions of optimisation problem have been proposed. A mono-objective optimisation problem P can be defined [147] as a quadruple (I, F, f, g), where:

- I is a set of instances.
- Given an instance $i \in I$, $F(i)$ is the set of feasible solutions.
- Given an instance i and a feasible solution x of i, $f(i, x)$ is the objective value.
- g is the goal function, and is either min or max.

Solving an instance i of a mono-objective optimisation problem involves finding its optimal solution, i.e. a feasible solution x with: $f(i, x) = g\{f(i, x')|x' \in F(i)\}$. In mono-objective optimisation problems, several optimal solutions might exist. Usually, the aim of the optimisation is to find any of them [75].
In the case of multi-objective optimisation problems, f is a vector function that comprises a set of scalar objectives [211]. The multiple objectives are usually conflicting, so a solution optimising every single objective might not exist [57, 58]. Without loss of generality, in the following description it is assumed that every scalar objective must be minimised. A multi-objective optimisation problem can be formally described as:

$$Optimize\ f(x) = (f_1(x), f_2(x), ..., f_k(x))\ \ subject\ to\ x \in F$$

where $f(x)$ is the objective vector, $f_i(x)$ is the i-th objective to be optimised, k is the number of objectives to optimise, x is the decision vector and F is the feasible



(a) Non-dominated solutions (b) $a \preceq b$, $a \preceq c$

Figure 1.1: Pareto Dominance Examples

region in the decision space. Given that in multi-objective optimisation, several objectives are simultaneously considered, the notion of optimal solution must be redefined. The most common applied definition of optimality is the one defined by Francis Ysidro Edgeworth [88], and generalised by Vilfredo Pareto [200]. Such a definition of optimality constitutes by itself the origin of research in multi-objective optimisation. In fact, the main aim of multi-objective optimisation is to identify the set of Pareto Optimal solutions. In order to formally define a Pareto Optimal solution, the concept of Pareto domination must be given:

Definition 1 *A decision vector x* ***dominates*** *the vector y ($x \preceq y$) if* $\forall i \in \{1, \ldots, k\} \quad f_i(x) \leq f_i(y) \wedge \exists i \in \{1, \ldots, k\} : \quad f_i(x) < f_i(y)$

That is, a solution x is said to dominate a solution y if and only if x performs better than y in at least one objective and at least as good as y in the rest. Figure 1.1 illustrates the concept of Pareto domination. It shows the objectives values of a set of solutions for an optimisation problem with two objectives. Minimisation is assumed in both objectives. In subfigure a, no one of the solutions is better than any other when both objectives are taken into consideration. Thus, there are no dominations among the solutions. In subfigure b, the solution a dominates both the solutions b and c. Both $f_1(a)$ and $f_2(a)$ are lower than the corresponding objective values of b and c. However, there is no domination between solutions b and c.
The definition of domination can be extended to sets of solutions. The set of solutions A is said to dominate the set of solution B, if every solution in the set B is dominated by at least one solution of the set A.

Definition 2 *A decision vector x* ***strongly dominates*** *the vector y ($x \prec y$) if solution x is strictly better than solution y in every objective, i.e. $\forall i \in \{1, \ldots, k\}\ \ f_i(x) < f_i(y)$*

In order to distinguish between both kinds of dominations, some authors refer to the domination $x \preceq y$ as weak domination. In this dissertation, the term domination always denotes the weak domination.

Definition 3 *A decision vector x is a* ***Pareto Optimum*** *if there is no solution $y \in F : y \preceq x$.*

These solutions are optimal in the sense that no other solutions in the search space are superior to them when all objectives are considered, i.e. a solution x is said to be Pareto Optimal, or non-dominated, if there not exist any solution y that performs better than x in at least one objective and at least as good as x in the rest. Usually, practitioners refer to the set of solutions obtained by an optimisation scheme, as the set of non-dominated solutions. In this case non-dominated solutions mean that the scheme has not been able to find any solution that dominates them. However, if every candidate solutions were explored, a solution which dominates some of them might be found. Some other related concepts are the Pareto Set and the Pareto Front.

Definition 4 *The* ***Pareto Set****, $\mathcal{P}^*$, for a given multi-objective optimisation problem P, and instance i, is the set of feasible solutions that are Pareto Optimum.*

It is important to note that the vectors in the set $\mathcal{P}^*$ are in the space of the variables. However, in order to identify them, their transformations to the space of the objectives are used.

Definition 5 *Given a problem P and instance i, whose Pareto Set is $\mathcal{P}^*$, the* ***Pareto Front****,* ***Pareto Optimal Front****, or* ***Pareto Frontier*** *is defined as $\mathcal{PF}^* = \{f(x),\ x \in \mathcal{P}^*\}$*

Thus, the Pareto Front of a problem is made up by applying the function f to each solution of the Pareto Set.

1.2 Optimisation Strategies

Several optimisation strategies have been designed with the aim of dealing with optimisation problems. Their corresponding algorithms can be classified in different groups in terms of different properties. This section shows a classification based

on the next properties of the algorithms: implementation, complexity, and design paradigms.

Classification by Implementation

The optimisation strategies can be categorised depending on several implementation details. Some of the most used are:

- *Programming paradigm*: there are several programming paradigms to develop algorithms. In *imperative programming* algorithms are described in terms of statements that change the program state. Imperative programs define sequences of commands that the computer must execute. In *functional programming* algorithms are expressed as the evaluation of mathematical functions. This kind of programming avoids the usage of states and mutable data. Thus, in contrast with imperative programming, in functional programming the results of a function only depends on its arguments. In *logic programming* algorithms are expressed as logical inference rules. These algorithms consist of two main components: logic and control. The logic expresses the axioms that may be used in the computation and the control determines the way in which deduction is applied to the axioms.

- *Recursive or iterative*: a recursive algorithm is one that invokes itself repeatedly until a certain final condition is satisfied, whereas iterative algorithms use repetitive constructs - like loops - and never invoke themselves.

- *Serial or parallel or distributed*: serial algorithms execute one instruction at a time, while parallel and distributed algorithms run on computer architectures where several processing units can work together. Thus, several sentences are executed at the same time. In order to afford the cooperation among processors, the algorithm is usually divided into different subtasks which can be distributed or solved by different units. The results are then gathered to yield the final solution of the algorithm.

- *Deterministic or non-deterministic*: deterministic algorithms solve the problem with exact decisions at every step of the algorithm. In this kind of approaches, randomness is not used. Therefore, given a fixed input, the same outcome is always obtained. Non-deterministic algorithms, also known as stochastic algorithms, introduce some decision steps based on randomness and probabilities. Therefore, this kind of algorithms does not guarantee the achievement of similar outcomes for an identical input in different executions of the approach.

Classification by Complexity

Algorithms can also be classified based on their complexity. The complexity of an algorithm is a function that expresses the amount of resources - usually time and memory - required by an algorithm to complete. Several notations for the complexity have been proposed. The O-notation is the dominant method used to express the complexity of algorithms [32]. This notation characterises procedures and algorithms according to their growth rates in the worst-case. Thus, a complexity $O(g)$ in the time, means that the execution time of the algorithm grows asymptotically no higher than g, being g a function which usually depends on the inputs of the algorithm. The complexity of the algorithm is useful to analyse the limitations of the designed approaches and to predict the resources required by the designed approaches to finalise.
Optimisation problems are also categorised using their complexity. Specifically, if a given problem may be solved to optimality by using multiple algorithms of different complexities, the one with the best worst-case is considered. Complexity classes collect the problems that can be solved using no more time than a specified amount. The most prominent complexity classes are P and NP. The class P refers to the problems solvable by some algorithm within a number of steps bounded by some fixed polynomial in the length of the input. The class NP refers to problems which are solvable by a non-deterministic touring machine in polynomial time. The complexity class P is contained in NP. The opposite is unknown [61]. The class NP contains many important problems, the hardest of which are called *NP-Complete* problems. A NP problem p is also a *NP-Complete* problem if and only if every other problem in NP can be transformed into p in polynomial time. A large number of optimisation problems are classified as *NP-Complete* problems [69]. Several of the optimisation problem addressed in this thesis are known to be *NP-Complete*.

Classification by Design Paradigm

Another classification criterion is based on the general methodology applied in the design of the algorithms. Many design paradigms have been proposed in the literature. Depending on the problem type and properties, on the aim of the optimisation, and on the size of the instances, some of the methodologies will be more appropriate than others [180]. Figure 1.2 shows some of the most used paradigms. A high-level classification of the paradigms divides them into two categories: exact and approximation approaches. Exact algorithms guarantee the achievement of optimal solutions. However, they usually require a large amount of resources, even for small instances. Thus, many real-world problems can not be tackled with exact approaches. Among the exact approaches is worth mentioning the next ones:

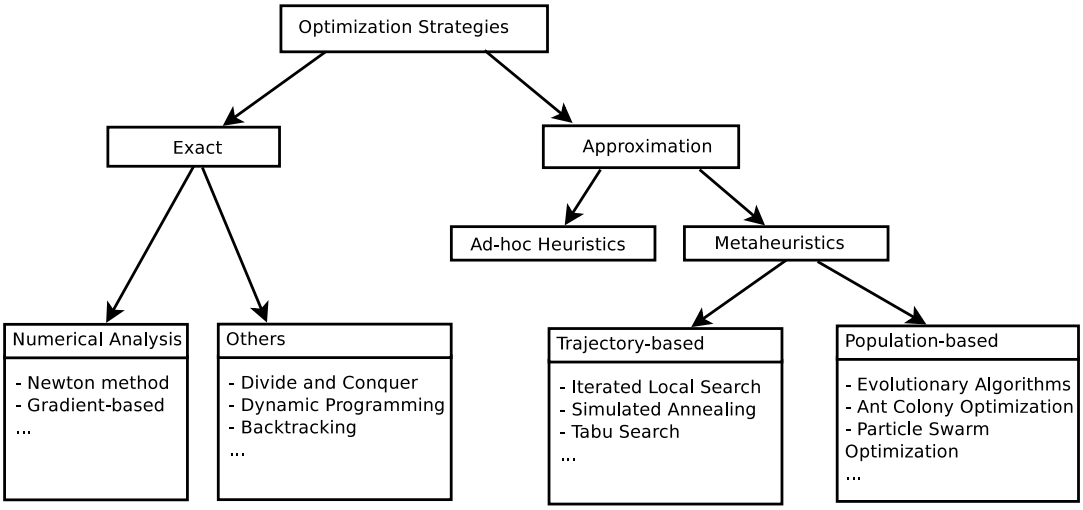


Figure 1.2: Classification of Optimisation Strategies

- *Numerical Analysis* [225]: this kind of method is based on the usage of numerical approximations to solve the problem. Such approximations can only be applied when some properties hold for the underlying optimisation function, so they are not general methods. Although the achievement of the optimal value is not ensured in every numerical method, generally the properties of the convergence of each numerical strategy are known. Thus, the achievement of an error lower than some fixed threshold might be ensured. Since other approximation methods do not ensure any properties regarding the convergence, in many classifications they are categorised as exact methods [163].

- *Divide and conquer* [151]: an instance of a problem is repeatedly reduced to a set of smaller subinstances or subproblems. When the subproblems are small enough, they are solved and the results are subsequently combined until a complete solution is achieved for the initial problem. Three generic computational operations can be identified in this model: split, compute, and join.

- *Dynamic programming* [23]: in many problems, the optimal solution can be constructed from optimal solutions to slightly small subproblems. In such cases, the problem usually exhibits the properties of overlapping subproblems, i.e. the same subproblems are used to solve several different problem instances [70]. In dynamic programming, the solution of each solved subproblem is stored in memory. Thus, the recomputing of solutions that have already been computed is avoided.

- *Enumeration techniques* [196]: these techniques are based on enumerating the different feasible solutions of the decision space, probably pruning some of them. The prune is done on the basis of a mathematical argument that guarantees that exploring the pruned zones is not required to obtain an optimal solution. Some of the most common algorithms in this category are exhaustive search, branch and bound, and backtracking with pruning.

Approximation approaches try to find reasonably good solutions reasonably fast. The requirements in time and memory of approximation algorithms are much lower than the requirements of exact approaches. Thus, by using approximation approaches, real-world instances of complex optimisation problems can be tackled. However, they do not guarantee the achievement of optimal solutions. Among the approximation algorithms, heuristics are very popular.

Definition 6 *A **heuristic** is a criterion, method, or principle for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal [202].*

The design of heuristics represents a compromise between two requirements: the need to make simple and low-cost methods and, at the same time, the desire to see them discriminate correctly between good and bad choices. The heuristic design is usually based on problem-dependent information. Heuristics can be used as a whole optimisation algorithm, or they can also be used inside exact search strategies as a way to recommend which of a set of possible solutions is to be examined next. Thus, they can be used as a way of speeding-up an exact approach.
Heuristics are usually based on specific knowledge of the problem. Therefore, its design and implementation is a hard task. Usually, they can not be easily adapted to other similar optimisation problems, or even they might be useful only for a subset of the instances of an optimisation problem. In order to improve the behaviour of heuristics, they can be integrated with other techniques, as re-starting and randomisation strategies. Also, by combining different heuristics, results can be improved. However, combining different simple heuristics is not an easy task, and it might yield to unsatisfactory results [111]. *Metaheuristics* appear as a class of modern heuristics whose main goal is to combine heuristics in a problem independent way with the aim of improving their performance [247].

Definition 7 *A **metaheuristic** is a heuristic method for solving a very general class of problems. It combines objective functions or heuristics in an abstract and hopefully efficient way, usually without considering deeper insight into their structure.*

Metaheuristics are defined as master strategies that guide and modify other heuristics with the aim of improving the solutions obtained by them. They are usually non-deterministic approaches whose main goal is to efficiently explore the search space in order to find near-optimal solutions. The design of metaheuristics is not problem-specific. However, they might make usage of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy. Most of the metaheuristics try to combine in an intelligent manner the processes of intensification and diversification [112]. In the intensification stage the search focuses on examining neighbours of elite solutions. The diversification stage encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those explored before. Compared to exact search methods, metaheuristics are not able to ensure a systematic exploration of the entire solution space. Instead, they attempt to examine the parts where "good" solutions may be found. Well-designed metaheuristics try to avoid being trapped in local optima or to repeatedly cycle through visited solutions. Moreover, given the limitation of resources, it is also important to provide a reasonable assurance that the search does not overlook promising regions.

A wide variety of metaheuristics have been proposed in the literature [29]. They range from simple local searches to complex learning procedures. Many of them are based on utilising statistics obtained from samples previously visited in the search space; others are based on some natural phenomenon or physical process models. For example, Simulated Annealing (SA) decides which solution candidate to be evaluated next according to the Boltzmann probability factor of atom configurations of solidifying metal melts, while Evolutionary Algorithms (EAs) mimic the behaviour of natural evolution and treat candidate solutions as individuals competing in a virtual environment. Other widely used metaheuristics are: Tabu Search (TS), Scatter Search (SS), Memetic Algorithms (MAs), Greedy Randomised Adaptive Search Procedure (GRASP), Variable Neighbourhood Search (VNS), Iterated Local Search (ILS), and Ant Colony Optimisation (ACO).

Several classifications of metaheuristics are seen to exist [50]. A very popular one makes a distinction between trajectory-based and population-based approaches [30]. The main difference between these two kinds of methods lies in the number of tentative solutions managed at each step of the algorithm. Trajectory-based approaches, also named single point search, maintain at any instant of the optimisation process a single solution x. At each step, a transformation over x is performed, generating a new candidate solution. The term trajectory refers to the fact that the search performed by these methods is characterised by a trajectory in the search space. A successor solution may or may not be a neighbour of the current solution. Population-based approaches maintain a pool with several candidate solutions in

every iteration of the algorithm. The final results depend on the way the population is manipulated. The basic actions are add or delete candidate solutions from this pool, as well as make transformations over a candidate solution.
Another popular classification distinguishes between metaheuristics for static optimisation and metaheuristics that have been adapted for dynamic optimisation. In static optimisation the function f does not depends on the time. Thus, the metaheuristic can assume that f only depends on the decision vector x, and therefore, exploring the same region several times in not necessary. However, in the dynamic case, the function f changes over the time. They are used to model time-varying systems. In this kind of optimisation problems, the quality of a decision vector depends on several factors that change over the time. In this thesis every tackled problem has static objectives.
A widely explored research topic in metaheuristics deals with the design of suitable stopping criteria. In order to avoid a waste of resources, an optimisation run should be terminated as soon as convergence has been obtained. However, detection of this condition is not a trivial task, and it depends on several features of the problem [253]. Since the set of candidate solutions is usually very large, metaheuristics are typically implemented so that they can also be interrupted after a user-specified budget. This budget is usually defined in terms of the maximum execution time, or in terms of maximum number of function evaluations which can be performed. Most of the experiments in this thesis have been performed using a fixed budget of resources as stopping criterion. However, in order to be able to analyse the convergence features of the analysed models, the way in which the quality of the solution has evolved during the executions has also been considered.

1.2.1 Parameter Setting

Many real-world optimisation problems are characterised by large and complex search spaces. In order to solve them, many difficulties have to be faced. When dealing with such difficult and large-size optimisation problems, metaheuristics are preferable to classical optimisation methods [231]. However, from the methodological side, the development of efficient metaheuristics results in a complex process. Moreover, it requires the setting of a large number of parameters and the decision between numerous search components at different design steps.
In general, in order to successfully apply a metaheuristic several components and parameters must be specified. For instance, EAs - which has been widely used in this research - are a class of algorithms based on the same generic framework whose details need to be specified to obtain a particular EA. It is customary to call these details the EA parameters. Among others parameters, in EAs [93], the mutation,

crossover, and selection operators, as well as several probabilities must be set up. The quality of the solutions highly depends on such components and parameterisations [8, 167]. Thus, it is very important to perform the parameterisation in a proper way.
Usually, when a novel problem is tackled, there is no a priori information of which metaheuristic is the most suitable one. Therefore, in order to obtain high-quality solutions, several metaheuristics, as well as several parameterisations for each of them must be tested. Thus, the effort required to obtain high-quality solutions, both in terms of user-effort and computational-effort, might be very large. The complexity and cost of the selection process highly hinders the usage of metaheuristics. The selection of the parameters can be done in several ways [18]:

- Checking in a systematic way some ranges of the parameter values and assessing the performance of each value.
- Based on the experiences reported in the literature for similar optimisation problems.
- Performing a theoretical analysis of the behaviour of the metaheuristic for determining the optimal parameter setting.
- Using "standard" values.

Although every option has reported good results in different fields, using standard values, and checking the parameters in a systematic way are the most extended approaches [156]. In order to reduce the waste of resources of a systematic testing of the ranges, and with the aim of minimising the user effort, several studies has analysed the automation of the parameterisation of EAs and other metaheuristics [219].
Setting strategies usually divide the parameters and components into two categories [219]. The first group contains the parameters in which a metric of distance can be established among the accepted values. For instance, considering an EA, the crossover probability belongs to this group. The second group contains parameters with a finite domain and no sensible distance metric. For instance, considering again an EA, the crossover operator belongs to this group. The reason to make the distinction between this two groups is that the first group has a structure where relations between different parameters can be exploited. Specifically, it is usually assumed that two parameters with a low distance, will behave in a similar way. However, in the second group there is no an exploitable structure. The parameters in the first group are usually named *numeric parameters*, while the ones in the second group are named *symbolic parameters*.

The setting strategies are commonly divided into two categories: parameter tuning and parameter control. In parameter tuning [219] the objective is to identify the best set of parameters for a given metaheuristic. Then, the algorithms are run using these parameterisations, which remain fixed during the complete run. In this regard, it is important to remark that, usually, the parameters of the metaheuristics interact in highly non-linear ways [156]. Moreover, there are usually a large amount of parameterisation choices, but only little knowledge about the effect of the parameters. Thus, the problem of finding the best set of parameters is very hard. The stochastic behaviour of metaheuristics also hinders the design of the tuning strategies. The main drawbacks of parameter tuning are the following:

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- The process of parameter tuning is time consuming, even if parameters are optimised one by one regardless of their interactions.
- For a given problem, the selected parameter values are not necessarily optimal for all the stages of the optimisation, even if the effort made for setting them was significant.

Several studies [91] have concluded that the usage of static set of parameters during a metaheuristic run seems to be inappropriate. The main drawback of parameter tuning is that there is no guarantee that a fixed set of parameters leads to optimal performance because the demands of optimisation algorithms change during an optimisation run from exploration in early stages to exploitation in later stages. Since different parameter settings are needed to emphasise either exploration or exploitation, it follows that optimal parameter settings might vary over time. In fact, it has been empirically and theoretically demonstrated that different values of parameters might be optimal at different stages of the optimisation [14, 223].
Several metaheuristics as Simulated Annealing and Evolution Strategies provide self-adaptive parameters to deal with the requirements of each optimisation stage. Since they have been of great value in several fields, it seems promising to incorporate these ideas into other metaheuristics. Moreover, it would be of a great value to integrate these ideas in a general way, i.e. not taking into account the meaning of the controlled parameters. In this way, a large number of metaheuristics might profit from the designed strategies.
Parameter control strategies allow changing the parameter values during the metaheuristic runs. Thus, the aim of parameter control is to design a control strategy that selects the parameters to use at each stage of the optimisation. Several strategies that adapt the parameters of the algorithms have been designed. Most of

them [223, 252] depend on the formulation of the metaheuristic, and on some properties of the controlled parameters, so they can not be applied in a straightforward manner to other metaheuristics and/or parameters. Hyperheuristics are popular general methods which can be applied to perform the parameter control. The main advantage of hyperheuristics is that they are independent of the metaheuristics and parameters that they control.
A hyperheuristic can be viewed as a heuristic that iteratively chooses between a set of given low-level (meta)-heuristics in order to solve an optimisation problem [40]. Hyperheuristics operate at a higher level of abstraction than traditional heuristics, because they have no knowledge about the problem domain. The motivation behind the approach is that, ideally, once a hyperheuristic algorithm has been developed, several problem domains and instances could be tackled by only replacing the low-level (meta)-heuristics. Thus, the aim of using a hyperheuristic is to raise the level of generality at which most current (meta)-heuristic systems operate. Since the main motivation of hyperheuristics is to design problem-independent strategies, a hyperheuristic is not concerned with solving a given problem directly as is the case of most heuristics. In fact, the search is on a (meta)-heuristic search space rather than a search space of potential problem solutions. The hyperheuristic solves the problem indirectly by recommending which solution method to apply at which stage of the solution process. Generally, the goal of raising the level of generality is achieved at the expense of reduced - but still acceptable - solution quality when compared to tailor-made (meta)-heuristic approaches.
The low-level approaches set might consist of a metaheuristic with several different configurations of its parameters. Therefore, a hyperheuristic can be also viewed as a method to implement the parameter control. However, they go further, because they might also make possible the integration and coordination of several different metaheuristics with different characteristics.

1.2.2 Stagnation Avoidance

In many problems or specific problem instances some metaheuristics may have a tendency to converge towards local optima or other suboptimal regions. The likelihood of this occurrence depends on the shape of the fitness landscape [47]. For this reason, several methods have been designed with the aim of dealing with local optima stagnation [111]. Some of the simplest techniques are based on performing a restart of the approach when stagnation is detected [171]. In other cases, a component which inserts randomness or noise in the search is used [54]. Maintaining some memory, with the aim of avoiding exploring the same regions several times, is also a typical approach [112]. Finally, population-based strategies intrinsically try

to maintain the diversity of a solution set. In such strategies by recombining the set of solutions, larger areas of the decision space might be explored.
In the particular case of EAs, premature convergence appears when the population reaches such a suboptimal state that the genetic operators can no longer produce offspring that outperform their parents [4]. The impact of premature convergence is quite similar to the problem of stagnating in a local optimum. Several specific measures have been analysed with the aim of avoiding premature convergence. Some of the most popular ones are incorporating incest prevention [96], increasing the mutation rate [209], generating random offspring [209], or using self-adaptive selection pressure schemes [4].
Another novel and promising choice is the usage of *multiobjectivisation*. Multiobjectivisation was introduced in [146] to refer to the reformulation of originally mono-objective problems as multi-objective ones. A multi-objective optimisation strategy must be applied to solve a problem that has been multiobjectivised. Multiobjectivisation changes the fitness landscape of the optimisation problem, so it can be useful to avoid premature convergence and/or local optima stagnation [121]. Consequently, multiobjectivisation might facilitate the resolution of the considered problem. However, it can also produce a harder problem [34]. Among the benefits of multiobjectivisation it is important to remark that it is independent of the considered optimisation strategy, i.e. several optimisation strategies might be applied with the same multiobjectivisation scheme.
Multiobjectivisation can be carried out following two general schemes. The first one is based on decomposing the original objective function, while the second one is based on adding new objective functions. The addition of alternative functions can be performed by considering problem-dependent or solely problem-independent information. In addition, some multiobjectivisation schemes require the specification of some additional parameters. This hinders the parameter setting of the approach. However, since multiobjectivisation has allowed the achievement of high-quality results in several problem domains [159, 188, 245], it constitutes a promising approach as a method to avoid local optima stagnation.

1.2.3 Performance Metrics

The usage of stochastic optimisation strategies implies that in each run results with different quality might be obtained. This highly hinders the process of measuring the performance of the involved strategies, so performing fair comparisons among such optimisation strategies is even more difficult. In this dissertation the Run-Length Distributions (RLDs) [98] have been one of the tools used to perform the comparisons among the applied strategies. RLDs, also named run-time distributions, or time-to-

target plots, display on the ordinate axis the success ratio and on the abscissa axis the running time. The success ratio is defined as the probability that an algorithm will find a solution at least as good as a given target fitness value within the given running time. RLDs were proposed in [98] and have been analysed and recommended by several researchers [128]. RLDs have also been used to assess the performance of parallel solvers [208]. In this regard, the relative performance between two different solvers (parallel or not) have been measured in this work considering the times or evaluations required by each one of them to attain a given success ratio and a given target quality value.
Moreover, in order to provide results with statistical confidence suitable statistical analyses must be performed. In this dissertation, the statistical comparisons have followed the guidelines presented in [80, 215]. First, a Kolmogorov-Smirnov test is performed in order to check whether the values of the results follow a normal (gaussian) distribution or not. If so, the Levene test checks for the homogeneity of the variances. If samples have equal variance (positive Levene test), an ANOVA test is done; otherwise a Welch test is performed. For non-gaussian distributions, the non-parametric Kruskal-Wallis test is used to compare the medians of the data. A confidence level of 95% has been considered in every performed statistical test, which means that the differences are unlikely to have occurred by chance with a probability of 95%.

Multi-objective Metrics

The main goal of multi-objective solvers is to approximate the Pareto set. As in mono-objective optimisation, the notion of performance involves maximising the quality of the achieved solutions, while minimising the time required to obtain such solutions. The outcome of multi-objective solvers is not a single solution, but a set of trade-offs. For these reasons, the definition of quality is substantially more complex than for mono-objective optimisation. In fact, the optimisation goal of multi-objective solvers involves multiple objectives [255]:

- The distance of the resulting non-dominated set to the Pareto Front should be minimised.
- A good distribution of the solutions found is desirable.
- The extent of the non-dominated front should be maximised.

Several methods to assess the quality of non-dominated sets have been proposed [94, 254]. Such methods can be classified into two main categories [144]: the ones based

on attainment functions [104], and the ones based on quantitative quality indicators. Additionally, a third category contains the metrics that compare pairs of non-dominated sets relatively to each other [254].

Definition 8 *An **attainment function** (α) is a mathematical function that models the outcome of a multi-objective optimiser. For each objective vector z, $\alpha(z)$ is the probability that the objective vector z is attained by the optimiser.*

The true attainment function is usually unknown. However, an approximation can be easily estimated empirically. Specifically, the attainment function can be estimated from a sample of r independent runs of an optimiser via the *empirical attainment function (eaf)*:

$$\alpha(z) \approx eaf(z) = \frac{1}{r} \times \sum_{i=1}^{r} I((A^i \preceq \{z\}) \vee (\{z\} \subseteq A^i))$$

where A^i is the i^{th} approximation set obtained by the optimiser, and I is the indicator function, which evaluates to one if its argument is true and to zero if its argument is false.
From a practical point of view, first, the solver is executed r times. Then, for each objective vector z, the number of generated sets that contain a solution z' equal to z or for which the relation $z' \preceq z$ holds is counted. Finally, such a number divided by the overall sample size is the desired probability. Attainment functions are very useful to visualise the quality of the obtained results. However, its usage with more than three objectives is very complex.
Based on the attainment functions, the attainment surfaces have been defined. An attainment surface of a given approximation set A is the union of all the tightest goals that are known to be attainable as a result of A. In some cases, it is interesting to analyse the set of vectors which are attained with some probability. The k%-attainment surface divides the objective space in two parts: the goals that have been attained and the goals that have not been attained with a frequency of at least k percent. A procedure to calculate them was proposed in [142]. Figure 1.3 shows three different attainment surfaces. They divide the objective space in several zones: the zone which has never been attained, the zones which have been attained with a frequency larger or equal than 10% and 50%, and the zone which has been attained in every execution.

Definition 9 *A **unary quality indicator** I is a mathematical function that quantify the quality of an approximation set. For an approximation set Ω, $I(\Omega)$, is a real number that represents the quality of Ω.*

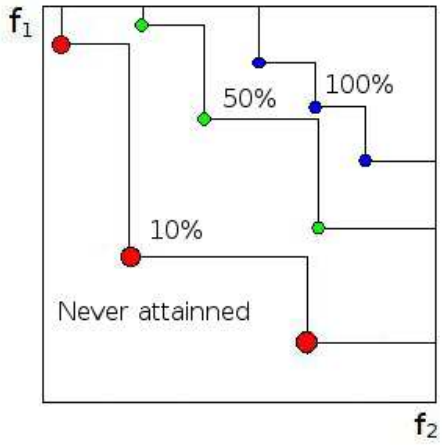


Figure 1.3: Attainment surfaces (10%, 50% and 100%)

The quantitative indicator approach summarises the outcome of a run as a number. The calculated number can then be used as a measure of the quality, so such a measure can then underpin statistical comparisons between the multi-objective solvers. The RLDs might also be applied, considering a fixed target value for the quality indicator.
Several quantitative quality indicators have been proposed [74]. Some of the best known are the error ratio [239], the generational distance [239], the spread [78], the hypervolume or S-metric [254], and the unary additive ϵ-indicator [259]. The error ratio and the generational distance evaluate the closeness of the obtained sets to the Pareto Optimal Front. The spread is a measure of the diversity of the non-dominated solutions. Finally, the hypervolume and the unary additive ϵ-indicator metrics evaluate at the same time the closeness and diversity of the obtained set. In the following lines a brief definition of each quality indicator is given. In them, the symbol S refers to the set of solutions which is being evaluated, while $\mathcal{PF}^*$ is used to denote the Pareto Front of the considered problem.

- The *error ratio* counts the number of solutions in S which are not member of $\mathcal{PF}^*$.
- The *generational distance* is the mean Euclidean distance between the solutions in S, and its closest solution in $\mathcal{PF}^*$.
- The *spread* measures the extension of the solutions in S.

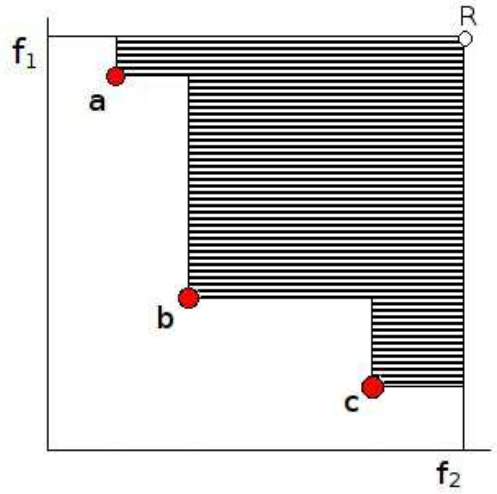


Figure 1.4: Hypervolume metric

- The *hypervolume* is the volume (in the objective space) covered by the solutions of S. The definition of a reference point (R) is required to calculate such a volume. Figure 1.4 illustrates the hypervolume metric for an optimisation problem with two objectives. In such a figure minimisation has been assumed. The hypervolume is the area of the hatched region.

- The unary additive ϵ-indicator metric measures the smallest amount, ϵ, that must be used to translate the set $\mathcal{PF}^*$ into S, i.e. it calculates the minimum summand ϵ to which each solutions from $\mathcal{PF}^*$ can be added in every objective so that the resulting approximation set is weakly dominated by S. In the cases where $\mathcal{PF}^*$ is not known a set containing the best known solutions might be used.

Definition 10 *A **binary quality indicator** I is a mathematical function that quantify the difference in quality between two sets of objective vectors. For the approximation sets A and B, $I(A, B)$ is a real number that represents the difference between the qualities of the considered sets.*

Several binary quality indicators have been proposed with the aim of comparing two sets relatively to each other [254]. Among them, some of the most extended are

approaches the coverage ($\mathcal{C}$) [254] , the coverage difference ($\mathcal{D}$) [254], and the binary additive ϵ-indicator [259]. In the following lines a brief definition of such binary quality indicators is given.
The *coverage* of a non-dominated set A, respect to a non-dominated set B ($\mathcal{C}(A, B)$) is the fraction of the solutions of B which are dominated by solutions in the set A. The coverage is calculated in the following way:

$$\mathcal{C}(A, B) = \frac{|\{b \in B / \exists a \in A : a \succeq b\}|}{|B|}$$

Thus, $\mathcal{C}(A, B) = 1$, means that every solution in the set B are dominated by solutions in the set A. Cases where $\mathcal{C}(A, B) = 0$ represents the situation when none of the solutions in B are dominated by solutions in A.
The *coverage difference* of a non-dominated set A, respect to a non-dominated set B ($\mathcal{D}(A, B)$) is the size of the objective space dominated by the set A, but not dominated by the set B.
The coverage difference is calculated in the following way:

$$\mathcal{D}(A, B) = hypervolume(A + B) - hypervolume(B)$$

In this research the recommendations given in [74, 143, 144, 259] have been followed. The comparisons among multi-objective solvers have been performed mainly in base of the hypervolume metric and attainment surfaces. The hypervolume has been used to assess the quality of a set because it is a Pareto-compliant metric, and because the knowledge of $\mathcal{PF}^*$ is not required. Moreover, since it does not use a reference set (it only use a reference point), results obtained by other approaches can be easily compared with the results presented in this dissertation. The same statistical tests than in the mono-objective cases have been used, but considering the hypervolume values, instead of the fitness values. Similarly, the RLDs have also been used.
Considering the conclusions drawn in [259], in the cases where it is stated that the solutions obtained by a scheme A are superior than the ones obtained by a scheme B in terms of the obtained hypervolume, it means that:

- The hypervolume values obtained by the method A are higher than the ones obtained the method B.
- Differences are statistically significant.
- Solutions obtained by the method A are not worse than the ones obtained by the method B in terms of dominances.

1.3 High Performance Computing

Most applications in scientific and engineering fields are computationally highly intensive. Therefore, in numerous scientific applications, the size of the problems and/or the amount of required computations implies the usage of highly optimised algorithms, and parallel systems. In the case of code optimisation, great improvements have been introduced in the last decades in the development of compilers. Thus, current compilers can perform several automatic optimisations which can lead to an impressive leverage over the performance. Moreover, much research has been conducted on algorithmic and data structures design [182]. However, the inherent complexity of problems and the high amount of data that must be managed in several fields still dictate the high computational effort associated with the solution of several problems. Using more powerful Central Processing Units (CPUs) and hardware devices helps to speed up the execution of the applications, but in this field there are also some limitations. The history of computing hardware, which involves a significant interest in increasing the component speeds, has been associated with Moore's law [185]. Since the invention of the integrated circuit in 1958, the number of transistors that can be placed inexpensively on an integrated circuit has increased exponentially, doubling approximately every two years. Such a statement is directly related to the increase in the speed and power of most electronic devices. The statement has held for about half of a century. However, since the Complementary Metal-Oxide-Semiconductor (CMOS) technology is nearer and nearer its physical limits, this miniaturisation can not continue forever.

High Performance Computing (HPC) [82] appears as an alternative, and refers to any computing resource that provides more computing power than is normally available. The goal is to achieve the maximum amount of computations in the minimum amount of time. For this purpose, the computing resources studied include the computers, networks, algorithms and environments necessary to make such systems usable. The concept of parallelism is highly related to high performance computing. Parallelism is not a new concept and was already extensively used far before the emergence of computer science [19]. The concept is quite simple. It is based on gathering several working units and making them collaborate to perform a given task. This partial definition is very broad. In fact, it encompasses all the computing parallel systems going from parallel machines to distributed clusters, while also including a sequential machine with pipelines. Parallelism initially invaded computers at the processor level under several aspects. The first one took place during the age of scalar processors, in the development of coprocessors taking in charge some specific tasks of the working unit. Other aspects have resided in the processor itself. For instance, the inclusion of more complex components in the processors such

as pipelines and multiple computation units has provided the ability of executing several instructions at the same time. Those forms of parallelism are hidden to the programmer. For this reason, these mechanisms are called intrinsic parallelism. In contrast, explicit parallelism is not hidden from the programmers. Thus, such systems require the development of specific parallel software.
Parallel computing techniques are especially applied to large and highly complex scientific applications [105]. In general, the most important reasons why parallel computing is of such interest in the research and scientific computing communities are: time saving, allows to afford larger problems, provides concurrency (i.e. it makes it possible to perform different tasks at the same time), allows to take advantage of non-local resources, and offers an alternative to the limitations of serial computing. Currently, there is neither a single kind of parallel system nor a single kind of parallel programming paradigm.
The aim of this section is threefold. The first goal is to present the most common kinds of parallel architectures which can be encountered throughout the world. The second goal is to provide a classification of the parallel programming models. The last goal is to present which are the most common metrics used to compare parallel approaches. The metrics for parallel approximation algorithms are also analysed.

1.3.1 Parallel Computer Architectures

Currently, there are many ways to build parallel systems. Therefore, the number of possible parallel configurations is huge. Depending of the problem to solve, some architectures provide better performance than others. Hence, it is important to have a deep insight of the currently used computer architectures. Several classification of the parallel computer architectures have been proposed [85, 201, 236]. The Flynn's taxonomy [101] is commonly accepted as a reference in the domain. This classification is based on the way of manipulating instruction and data streams. It comprises four main architectural classes:

- Single Instruction Single Data (SISD): these are the conventional systems that contain one CPU and hence can accommodate one instruction stream that is executed serially over a single data.

- Single Instruction Multiple Data (SIMD): corresponds to systems that have several processing units that may execute the same instruction on different data in lock-step. Therefore, a single instruction manipulates many data items in parallel. They often have a large number of processing units, ranging usually from 1024 to 16384. The set of processing units is known as vector processor.

- Multiple Instruction Single Data (MISD): theoretically in these types of machines multiple instructions should act on a single stream of data. It is the only class that has not yet led to real implementations. It seems that the range of applications corresponding to this particular architecture is quite reduced.

- Multiple Instruction Multiple Data (MIMD): are the systems capable of executing multiple instructions on different data at the same time. MIMD systems may run many different subtasks in parallel in order to shorten the time for the main task to be executed. There is a large variety of MIMD systems and especially in this class the Flynn taxonomy proves to be not fully adequate [236]. Therefore, inside this class other subclassifications have been proposed. They typically distinguish between MIMD systems with shared memory, and MIMD systems with distributed memory.

Another typical classification of the architectures is based on the radius of the system, i.e. the physical distance between the processing units. The classification based on the radius better reflects the evolution trends:

- Uniprocessor Machines: mainly representing the SISD machines.

- Parallel Machines: built as a single machine containing several processing units. They include SIMD and MIMD architectures and some combinations of them.

- Local Clusters: collections of independent computers gathered in the same place and connected via a local network. Although they are intrinsically MIMD oriented, SIMD machines might be used at the node level.

- Distributed Clusters/grids/cloud: collections of local clusters, or any other computing facilities, scattered all around the world and linked together via the Internet or other non-dedicated networks. As local clusters, they mainly follow the MIMD model.

Parallel Machines with Shared Memory

Shared memory computers provide all the processors with the ability to access all memory as a global address space, i.e. multiple processors can operate independently but they share the same memory resources, as shown in Figure 1.5. Changes in a memory location performed by one processor are visible to all the other processors. The great advantage of such systems is that they neither require data distributions

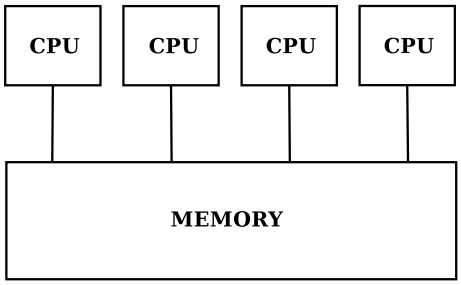


Figure 1.5: Shared memory architecture

over the processors nor data messages between them. Communications required between the processors are usually performed via the shared memory and can, thus, be very fast. However, the centralisation of the memory also presents some drawbacks. First, it implies a very high memory bandwidth. Thus, in order to avoid bottlenecks, cutting-edge technology must be used in the design of the memory. Moreover, the concurrent accesses may lead to incoherent results of the running application if it is not carefully managed. When many processors are integrated, the logic added with the aim of ensuring the *cache coherency* might highly reduce the performance of the system. Thus, the number of processors which can be integrated in a shared memory machine is not as high as in the machines with distributed memory.
The usage of Non-Uniform Memory Access (NUMA) is very typical in the parallel machines with shared memory. In such cases, the memory access time depends on the memory location relative to a processor. By using NUMA, the cost of the memory is reduced, and more processors can be added in a single machine. However, the design of the parallel software is more complex because it must take into consideration that the access time to the memory is not uniform.

Parallel Machines with Distributed Memory

As shown in Figure 1.6, processors in distributed memory systems have their own local memory, i.e. memory addresses of one processor do not map to another processor, so there is no concept of global address space across all processors. Since each processor has its own local memory, they all operate independently. Changes made locally by one processor have no effect on the memory of other processors. In this architecture, the processors are linked together by an interconnection network. Therefore, when a processor needs access to data in another processor, such a data must be sent across the network. It is usually the task of the programmer to explicitly define how and when data must be communicated.

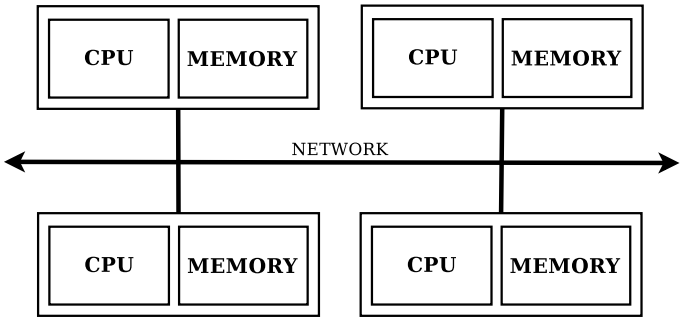


Figure 1.6: Distributed memory architecture

The main advantage of using distributed memory is its scalability. In particular, the concurrent memory accesses are no longer an issue because only one processor accesses one memory bank. For the same reason, the problem of the memory bandwidth is less critical. Hence, those systems are intrinsically better suited to include a very large number of processors. However, they also present some drawbacks. The most obvious one is the necessity of explicitly distributing the data among the processors. This usually complicates the development of parallel software. Moreover, since many applications require lots of communication, a high performance network might be required. The network is also required for exchanging information to control the applications. Therefore, the performance of the interconnection network might be a critical issue to ensure a good performance of the system.

Hybrid Distributed-Shared Memory

In general, the largest and fastest computers in the world employ a combination of both shared and distributed memory architectures. Several shared memory machines - with a set of CPUs and a single shared memory - are interconnected through a network, as depicted in Figure 1.7. Then, it is required to move data from one shared memory machine to another.

1.3.2 Trends of used Architectures

The previous classicisation has depicted the general schemes that have arisen since the beginning of parallelism. This section is devoted to describe the currently most extended configurations. In this regard, the number of architectures which incorporates multi-core processors has highly increased in the last years. Even computers not devoted to high performance computing usually consist of several processing

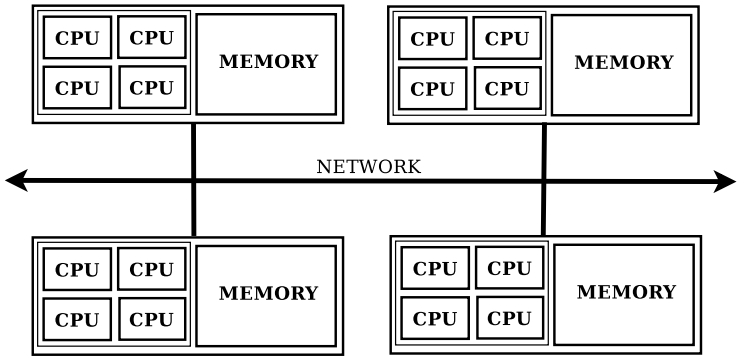


Figure 1.7: Hybrid distributed-shared memory architecture

units. The most extended multi-core processors usually incorporate between 2 to 12 cores. The cores usually make use of shared memory. Also, there is some tendency to incorporate several processors inside the same motherboard. Several machines with up to four processors in a single motherboard are available. However, the price of this kind of machines is usually much higher, so they are not so extended in the non-HPC environments.
When a large number of processing units is required, the most typical approach is to mix shared and distributed memory in a single environment. This is typically made with racks that contain node cards containing multi-core chips. The implementations are of two sorts. The commonly named Beowulf approach involves using off-the-shelf hardware for the nodes and networks. They are also named Networks of Workstationss (NOWs), or Clusters of Workstationss (COWs). The latter uses specific integration facilities with optimised network and software environment. The first approach is mostly used for creating medium-sized machines, while the last approach is used by the most powerful parallel machines developed up to this date.

1.3.3 Parallel Programming Models

Once we know that many types of parallel architectures are available, if we want to process programs and data faster, we must switch from hardware architecture to a software design that makes concurrent use of multiple processors. Traditionally, software has been written for serial computation, i.e. to be executed on a single computer within a single processing unit. In serial processing, programs consist of a set of instructions which are executed one after the other, since only a single instruction can be executed at any instant of time. On the contrary, parallel computing techniques are based on the simultaneous usage of multiple computing resources for

solving a given problem [28]. The parallel software developers must contend with problems not encountered in sequential programming, as communication, synchronisation, data partitioning and distribution, load-balancing, fault-tolerance, heterogeneity, deadlocks, and race conditions. Thus, the necessity of convenient parallel programming models as well as parallel libraries and compilers is obvious.
A parallel programming model is a concept that enables the expression of parallel programs. This allows executing a parallel program in several different architectures. The implementation of a programming model can take several forms such as libraries invoked from traditional sequential languages, language extensions, or complete new execution models. Classifications of parallel programming models can be divided broadly into two areas: by process interaction and by problem decomposition.
Based on the process interactions, i.e. the mechanisms by which parallel processes are able to communicate with each other, the next models are seen to exist:

- Shared memory: parallel tasks share a global address space which they read and write to asynchronously. This requires protection mechanisms such as locks and semaphores to control the concurrent accesses. The most prominent examples of the shared memory programming model are Open Multi-Processing (OpenMP) [52, 53, 198] and *POSIX Threads* [45].

- Distributed memory: parallel tasks exchange data through passing messages to one another. These communications can be asynchronous or synchronous. Message Passing Interface (MPI) [177, 199, 221] and Parallel Virtual Machine (PVM) [83] are the most used libraries that implement the distributed memory programming model.

- Implicit: in the implicit programming model, no process interaction is visible to the programmer, instead the compiler and/or runtime is responsible for performing the interactions.

This classification is highly related to the aforementioned architecture classification. However, note that they are independent, i.e. a program expressed in a given programming model, might be executed in several different parallel architectures. For instance, a program developed with MPI can usually be executed in a parallel architecture with shared memory. Also, a program developed with OpenMP might be executed in a distributed machine. Usually, this is done by emulating the shared memory with some external libraries. Thus, modifying the original program is not required.
The classification based on the problem decomposition is also named programming paradigm. There are also several classifications based on the performed decom-

positions [122, 250]. Based on this classification the next paradigms are usually identified:

- Task-Farming: this paradigm consists of two types of entities: a master and multiple workers. First, the master decomposes the problem into small tasks, and distributes them among the workers. Then, the workers execute each assigned task, and send the results to the master. Finally, the master gathers the results, and combines them to constitute the problem solution. Usually, this paradigm is combined with a dynamic load-balancing scheme. This allows dealing with processors that have different loads and with heterogeneous systems. The main drawback of this paradigm is its scalability. When a high amount of processors are used, the centralised control of the master process can become a bottleneck to the applications.

- Single-Program Multiple-Data (SPMD): this paradigm is one of the most commonly used paradigm. Each process execute the same code but on different part of the data. The main idea is to split the data among the available processors, but without the interaction of a master process. Usually some processors require data computed by other processors. Thus, depending on the problem, a neighbourhood might be defined. Then, processors communicate from time to time with their corresponding neighbours. Moreover, some global synchronisation may also be required.

- Data Pipelining: this paradigm is based on decomposing the tasks of the algorithm into several subtasks that can be executed concurrently. The efficiency of this paradigm depends on the ability of properly balancing the load across the stages of the pipeline. The communication pattern is usually very simple because data flows only between adjacent stages.

- Parallel Divide and Conquer: this paradigm is an extension of the well-known sequential paradigm Divide and Conquer. An instance of a problem is repeatedly reduced to a set of smaller subinstances or subproblems. When the subproblems are small enough, they are solved and the results are subsequently combined until a complete solution is achieved for the initial problem. Since each subproblem is independent, they might be assigned to different processing units. Three generic computational operations can be identified in this model: split, compute, and join. Depending on the computational cost associated to each one, different algorithms have been proposed.

- Speculative Parallelism: in this paradigm some processors execute some parts of the code speculating with the results that other procedures might obtain. In

some cases their assumptions are invalid, so computed data must be discarded. However, when their assumptions turn out to be correct, the performed computation might improve the overall performance. This paradigm is usually applied when the other paradigms can not be used.

- Hybrid paradigms: several parallel algorithms need to mix elements of different paradigms. In such cases, it is said that they follow a hybrid paradigm. For instance, there are several cases, where data and task parallelism is simultaneously applied.

- Other paradigms: the previous paradigms are suitable for exact algorithms. However, in the cases of approximation approaches, the parallel strategies might be completely different from the sequential approaches. In metaheuristics, for instance, it is typical to propose parallel approaches that change the way in which the search space is explored. For this reason, parallel metaheuristics are usually classified using different paradigms. Nevertheless, they usually make use of some of the ideas behind the here presented paradigms.

1.3.4 Metrics in Parallel Systems

Efficient parallel algorithms are required to profit from the usage of parallel systems. In this regard, for a given problem, the optimum parallel algorithm may be radically different from the optimum serial algorithm. The design goal of any parallel algorithm is to solve the problem minimising the required time. This is usually achieved by dividing the global task into independent sub-tasks, or tasks that require little synchronisation and communication. In general, efficient parallel algorithms result from the efficient use of process resources and the maximisation of the computation-communication ratio. In order to measure the performance or efficiency of parallel programs, several metrics have been proposed [243]. Some of the most common and widely used metrics are presented in this section.

Speedup

The term *speedup* is defined as the ratio of the time required to complete the process with the fastest serial algorithm using one processor to the time required to complete the same process with the parallel algorithm using p processors. Since the fastest serial algorithm is not usually known, this definition is relaxed, and the speedup is calculated taking as reference a given sequential approach. Thus, being T_1 the time invested by the sequential approach, and T_p the time invested by a parallel algorithm using p processors, speedup is calculated as:

$$S_p = \frac{T_1}{T_p} \tag{1.1}$$

Amdahl's law establishes that the *speedup* obtainable from a program is determined by the fraction of the program code that can be parallelised (P):

$$speedup = \frac{1}{1-P} \tag{1.2}$$

If there are no code sections that can be parallelised ($P = 0$), the achievable speedup is 1, i.e. it is not possible to speed up the program execution. If all the code can be parallelised ($P = 1$), then, theoretically, the speedup is infinite. If 50% of the code can be parallelised ($P = 0.5$), a maximum speedup of 2 might be obtained, meaning the execution is twice as fast.
The speedup also depends on the number of processors (p) that take part in the execution. Being S the fraction of code that is serial, the maximum speedup that can be achieved by a computer with p processors is defined as:

$$speedup \leq \frac{1}{S + \frac{P}{p}} = \frac{1}{S + \frac{1-S}{p}} \tag{1.3}$$

This formula clearly shows the limits that affect scalability in parallel programs. Increasing the size of the sequential part of the problem quickly causes the speedup to saturate, so when designing efficient parallel algorithms, sequential operations must be reduced, thus minimising the idle time of each processor.

Efficiency

In general, only ideal parallel systems can achieve a speedup of p for a *p-processor* system. This implies that the fraction of serial operations is 0. In practice, the ideal case can not be achieved since processors can not devote all of their time to computing the problem. There are overheads embedded in parallel programs such as inter-processor communication, data communication, synchronisation, etc. Efficiency is then proposed as a measure of the fraction of time for which processors are usefully employed:

$$efficiency = \frac{speedup}{p} \tag{1.4}$$

In the ideal case, speedup is p, so the efficiency is 1. In practice, speedup usually is less than p and so efficiency is a value between 0 and 1. Since efficiency is proportional to the speedup, it also decreases quickly as S increases.

Scalability

Usually, the speedup does not increase linearly when the number of processors increases. A constant speedup tends to be achieved as overheads due to communication, synchronisation, etc., increase. On the other hand, in many algorithms an increase in problem size yields to obtain a higher speedup and efficiency for the same number of processors. These two phenomena are common to many parallel systems. Scalability analyses examine the behaviour of parallel algorithms with different number of processors, and different problem sizes. An ideal scalable parallel system maintains efficiency as the number of processors increases under the condition that the problem size is also increased.

Parallelism and Approximation Approaches

Previous metrics are very well suited for exact and deterministic approaches. However, the usage of approximation non-deterministic algorithms highly hinders the analysis of the performance of the approaches. For instance, a parallel stochastic approach might speed up the achievement of high quality solutions in most of its executions, but in other ones, it could suffer from stagnation and behave even worse than a sequential approach. Moreover, since the sequential algorithms are also stochastic, the analysis is even more difficult. For this reason, the analysis of the performance of parallel stochastic schemes usually combine the previous metrics, with several graphics, statistical analysis, and alternative metrics that allow to have a better understanding of the approach.

Considering the stochastic behaviour of approximation methods, it makes no sense to evaluate the speedup considering solely the time used to complete an execution. RLDs have been used to assess the performance of parallel solvers [208]. In this regard, the speedup of a parallel solver with respect to a sequential approach has been calculated in this dissertation in the following way. First, a desired quality level is fixed. Such a quality level might be a fitness value for the mono-objective cases, or some metric value for the multi-objective cases. Then, the sequential and parallel solvers are executed several times, fixing as stopping criterion the achievement of such a quality level. The times required by the sequential model (T_1) and by the parallel model (T_p) to reaching a fixed success ratio are calculated. Finally, the speedup is calculated using equation 1.1.

1.4 Research Questions

Given the heuristic nature of metaheuristics, and the broad amount of fields in which metaheuristics have been applied, there are a large amount of open research questions for its optimal application. In [156] some of the open research topics regarding EAs have been enumerated. Most of these topics are very general, so they are open research questions not only in the field of EAs, but also in the more general field of metaheuristics. The majority of these topics are related with the robustness of metaheuristics, with its optimal usage, and with the ways in which stagnation might be avoided. These topics are closely related with the main difficulties that practitioners must face up to when applying metaheuristics to large and complex optimisation problems. In fact, one of the main drawbacks of metaheuristics is the difficulty associated to its parameterisation. The process of making the parameter setting of metaheuristics usually takes much user and computational effort [156]. However, since the performance of metaheuristics highly depends on its parameterisation, it is important to perform it in a suitable way.
Another currently active research topic is concerned with the usage of *multiobjectivisation* as a way of improving the behaviour of metaheuristics. The term *multiobjectivisation* was introduced in [146] to refer to the reformulation of originally mono-objective problems as multi-objective ones. This topic is highly related with the metaheuristic parameterisation, and with the optimal usage of metaheuristics. In fact, since there are several ways to multiobjectivise a problem, the way in which multiobjectivisation is performed might be considered as an extra parameter of the approach. Moreover, a multiobjectivisation scheme might define its own parameters. Thus, when multiobjectivisation is incorporated into a particular approach, properly tuning the algorithm can be even more time-consuming. In addition, the usage of multiobjectivisation changes the fitness landscape of the problem, so it can be useful to avoid stagnation in non-optimal regions.
Finally, the proper usage of metaheuristics in parallel architectures [7] is also an open research question. There are several parallel models, but up to now, no one has demonstrated to be superior to the others. The recent proliferation of parallel processing technologies, even in the non-HPC environments, makes this topic even more important.
Given the close relation among the aforementioned topics they have been analysed together. The main motivation of this research has been to develop a parallel model which can facilitate the usage of metaheuristics, and to solve practical applications with such a proposal. The main research questions addressed in this research are:

- Is it desirable to dynamically change the parameters values during the metaheuristic runs?

- Can these changes be performed by a general technique, or is it required to use information of the baseline metaheuristic?

- Is it necessary to incorporate problem-dependent information in order to perform the parameter control?

- How can parallel models and parameter control techniques be integrated?

- Can multiobjectivisation techniques be integrated in this model as if they were an extra parameter of the metaheuristic?

- Can these techniques be successfully applied to practical optimisation problems that have been analysed with many other metaheuristic approaches?

Some of the previous questions have been in the mind of metaheuristics practitioners for many years. In fact, many research papers have been published about the addressed topics. They are very important topics because giving answer to them would greatly facilitate the usage of metaheuristics for new users, and its application in the industry for novel and motivating optimisation problems. However, they are so broad topics, that it is expected that they will remain as open research topics for many years. The aim of the research is to improve the state of the art of the aforementioned research questions.

1.5 Contributions

The current thesis presents new methods and empirical results which address the questions discussed in the previous section. The specific contributions comprise:

- A set of new hyperheuristics for multi-point schemes has been designed. Such metaheuristics are based on using novel scoring and selection methods. The methods can be applied both for mono-objective and multi-objective optimisation problems. By using this model, metaheuristic practitioners do not need to manually test a large number of metaheuristics and parameterisations for discovering the proper algorithms to use. Instead, they can define the set of configurations which must be tested, and the model tries to automatically detect the best-behaved ones, in order to grant more resources to them. The model allows not only using different parameterisations of the same metaheuristic, but also to integrate different metaheuristics.

- A new parallel model which hybridises the island-based model and the designed hyperheuristics has been proposed. This model allows performing the parameter control of metaheuristics in a parallel way. Moreover, it goes further, allowing the integration of different metaheuristics in a general way. The cooperation among the metaheuristics is performed following the same schemes that in island-based approaches. The model has been used to solve both mono-objective and multi-objective optimisation problems. The validation has been performed by using well known mono-objective and multi-objective optimisation problems.

- Several novel multiobjectivisation techniques have been designed. Such techniques have made possible to avoid premature convergence and to speed up the convergence of EAs to high quality solutions with several optimisation problems. In order to use the designed multiobjectivisations, additional parameters must be set up. For this reason, such multiobjectivisations have also been integrated with the designed hyperheuristics. The hyperheuristic has been able to automatically adapt the parameters, obtaining better results than with any fixed value.

- Several computationally expensive and real-world problems have also been tackled in this thesis. In these cases, the aim has been two-fold. On the one hand, the validation of the hyperheuristics was an objective. On the other hand, the resolution of such problems was important because of its practical implications. For this reason, in some of them several novel genetic operators, neighbourhood definitions, and optimisation methods specifically designed for such problems have been proposed. In every case, they were integrated with the designed parallel approaches. The obtained results have been compared with the best published results for each of them. The usage of the parallel approach to solve well-known computationally demanding problems have been useful to draw more general conclusions about the benefits and drawbacks of the proposal. The following problems have been addressed:

  - Antenna Positioning Problem (APP) is an NP-Complete problem which arises in the engineering of mobile telecommunication networks [178]. APP is defined as the problem of identifying the infrastructures required to establish a wireless network. Some novel metaheuristics and genetic operators have been defined. The parallel approach has been able to speedup the achievement of high-quality solutions. In addition, it has facilitated the usage of the new designed metaheuristics.

  - Frequency Assignment Problem (FAP) is a well-known combinatorial optimisation problem of great importance in the telecommunication area. It is one of the key issues in the design of Global System for Mobile Communications (GSM) networks. The main aim of FAP is to assign the set of frequencies that must be used in the different antennas of the network, in order to minimise the quality loss of signal. For this problem a novel local search neighbourhood has been defined. The usage of the local search has provided many benefits both in terms of time saving, and solution quality. In combination with the parallel designed approach, and with the multiobjectivisation schemes, it has been able to achieve the currently best published network configurations for two real-world instances.
  - Optimisation of the broadcast operation in Mobile Ad-Hoc Networkss (MANETs). MANETs are fluctuating, self-configuring networks of mobile hosts, called *nodes* or *devices*, connected by wireless links. The broadcast operation is very important in this kind of networks. Several algorithms that perform the broadcast operation have been proposed in the literature. The parallel model has been used to optimise the operation of the broadcast algorithm named Delayed Flooding with Cumulative Neighbourhood (DFCN).
  - Two-Dimensional Packing Problem (2DPP) is a variant of a packing problem proposed in the Genetic and Evolutionary Computation Conference (GECCO) 2008 competition session. The proposed problem definition is different from traditional packing problems. In this case, the aim was to validate the parallel approach, and not to solve a packing problem. Therefore, the proposals have not been adapted to traditional packing problems. A novel local search neighbourhood definition was proposed. The performance of several multiobjectivisation schemes has also been analysed. The parallel approach combined with the defined metaheuristics has provided the currently best published results.

- A tool named Metaheuristic-based Extensible Tool for Cooperative Optimisation (METCO) [152] that allows executing the models developed in this research, as well as many other approaches designed by other researchers has been developed. The tool allows executing both sequential and parallel models. The functionalities of the tool can be extended through the definition of plugins, while the kind of execution to perform can be specified by using a configuration file. This means that it is not required to know the internals of the tool to use it. Finally, it is worthy to mention that several researchers of different universities are currently using the developed tool.

1.6 Overview

This dissertation has been divided in four parts:

- Part I: Fundamentals and Backgrounds

  This part is devoted to introduce the main concepts used throughout the dissertation, and to present the main contributions of the research. Chapter 2 presents the set of metaheuristics that have been used in this work. The main recent developments that are highly related to the models designed in this research are presented in Chapter 3. The main addressed topics are hyper-heuristics and multiobjectivisation.

- Part II: Problem-Independent Proposals and Validation

  This part contains the general algorithmic models designed in this research and its validation with well-known optimisation problems. Chapter 4 is devoted to describe the designed hyperheuristics and their parallelisations. In addition, the new designed multiobjectivisation schemes are explained. The validation of such schemes is presented in Chapter 5.

- Part III: Practical Applications

  This part presents the schemes that have been designed for tackling some practical and complex optimisation problems. The way of integrating such schemes with the problem-independent proposals is also described. Chapter 6 describes the schemes and results for three optimisation problems that arise in the communication field. The schemes and results obtained for a packing problem are presented in Chapter 7.

- Part IV: Conclusions

  This part is devoted to discuss the main conclusions drawn in the current research. In addition, some lines of future work are summarised.

Finally, an appendix that contains the set of publications which have been produced as part of the research has been included.

Chapter
2

Metaheuristics

This chapter is devoted to describe the set of metaheuristics that has been applied in this research. The main models for designing parallel metaheuristics are also presented. In some cases the metaheuristics have been used independently of the algorithmic novelties proposed in this thesis, i.e., the parallel hyperheuristics and adaptive multiobjectivisations. In such cases the main aim has been to solve a practical optimisation problem with the most adequate strategies, so considering other schemes was mandatory. Anyway, every addressed problem has also been tackled with the hyperheuristics and multiobjectivisations, so using such metaheuristics has been very useful to validate the novel proposals. The applied metaheuristics comprise a set of well known mono-objective and multi-objective approaches. In addition to the schemes described in this chapter, some other novel approaches that incorporate minor variations to well-known algorithms, and several hybrid approaches have also been used to face specific difficulties of some practical applications. They are described in the chapters of the corresponding practical optimisation problems.

2.1 Mono-Objective Metaheuristics

2.1.1 Evolutionary Algorithms

Evolutionary computation [93] is a special brand of computing which draws its inspiration from natural evolutionary processes. EAs are a subset of evolutionary computation techniques which are inspired by biological evolution. In natural evolution, a given environment is filled with a population of individuals striving to survive and reproduce. The fitness of these individuals is determined by the environment and relates to how well they succeed in achieving their goals. In a macroscopic view

Algorithm 1 Evolutionary Algorithm

1: **Initialisation**: Generate an initial population with N individuals
2: **Evaluation**: Evaluate all individuals in the population
3: **while** (not stopping criterion) **do**
4: **Mating selection**: select parents to generate the offspring
5: **Variation**: Apply genetic operators to the mating pool to create a child population
6: **Evaluation**: Evaluate the child population
7: **Survivor selection**: Select individuals for the next generation
8: **end while**

of evolution, natural selection plays a central role. Given an environment that can only host a limited number of individuals, and the basic instinct of individuals to reproduce, selection becomes inevitable if the population size is not grown exponentially. Natural selection favours those individuals that compete for the available resources most effectively, i.e. those that are best adapted to the environmental conditions. This phenomenon is also known as *survival of the fittest*. Competition-based selection is one of the two cornerstones of evolutionary progress.

The other primary force results from phenotypic variations among members of the population. Phenotypic traits are those behavioural and physical features of an individual that directly affect its response to the environment, thus determining its fitness. Each individual represents a combination of phenotype traits that is evaluated by the environment. If it is evaluated favourably, then it is propagated via the individual's offspring; otherwise, it is discarded by dying without offspring. Usually, small random variations in phenotypic traits occur during reproduction from generation to generation. Through these variations, new combinations of traits occur and are evaluated. The best ones survive and reproduce, thus evolution progresses. From a microscopic view of natural evolution, each individual is considered as a dual entity: its phenotypic properties (outside) which are represented at a low genotypic level (inside), i.e. an individual's genotype encodes its phenotype.

EAs belong to the group of population-based metaheuristics. There are many different variants of EAs but the underlying idea to all of these techniques is the same. Their basic generic framework is shown in Algorithm 1. First, an initial population with N individuals is created (line 1) and evaluated (line 2). Then, until the stopping criterion is reached (line 3) a set of steps are repeated. Such a set of steps is usually referred to as a generation of the EA. The steps are the following. First, the mating selection operator selects the parents among the members of the population (line 4). Such members constitute the mating pool. Then, a variation stage (line 5) is applied

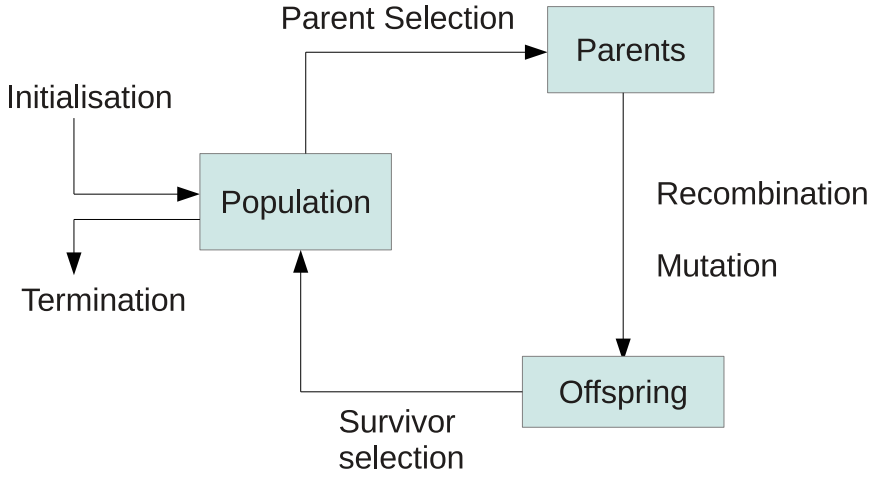


Figure 2.1: Flow-chart of an EA

to the mating pool. The most common operators are the crossover and mutation. Such step creates the offspring pool, which is then evaluated (line 6). Finally, the survivor selection scheme (line 7) selects the individuals that will constitute the new population. A flow-chart [93] of the scheme is shown in Figure 2.1.
In order to obtain a particular configuration of an EA several details have to be specified. Consequently, several variants of EAs are seen to exist. Specifically, the following components must be selected:

- Method to generate the initial population.
- Selection mechanisms: mating and survivor selection.
- Variation scheme: crossover and mutation operators.
- Encoding of the individuals, and its transformation to the phenotype.

In addition, some parameters must also be set up. The most common ones are the mutation and crossover probabilities, and the population size. Some specific flavours of EAs might have additional parameters.
In this research two of the EA components have remained fixed in every experiment. In the case of the generation of the initial population, a method that assigns random values to each gene has always been considered. The random assignments have been based on using a uniform distribution among the accepted gene values. The mating selection has been based on the well-known binary tournament selection.

In the case of the remaining components, several choices have been tested. Regarding the encoding of individuals the following choices have been analysed: encoding based on real-values, binary string encoding, and two-dimensional chromosomes.
The following survivor selection mechanisms have been used:

- Steady-State Selection (SS-S): In each generation, one offspring is generated. If it is better than any of the individuals of the population, the worst of them is replaced by this new offspring.
- Generational with Elitism Selection (GEN-S): In each generation, $N-1$ individuals are generated, being N the population size. All parents, except the fittest one, are discarded, and they are replaced by the generated offspring.
- Replace Worst Selection (RW-S): In each generation, N individuals are generated. The N fittest individuals, among parents and the new generated ones, are selected to survive.

The following mutation operators have been tested:

- Polynomial Mutation (PM) [77]: The selected gene values are changed to a neighbouring value using a polynomial distribution that has its mean value at the current value. The variance is a function of the distribution index η. In every experiment $\eta = 20$ has been used. This mutation is used with real-valued encodings.
- Uniform Mutation (UM) [93]: This operator replaces the values of the chosen genes with random values selected between the specified upper and lower bounds of such genes, following a uniform distribution. This mutation is used with real-valued encodings.
- Flip mutation [93]: The values of the chosen genes are flipped. This mutation is used with the binary string encoding, so the chosen genes are inverted from 1 to 0 or from 0 to 1.
- Specific mutation: In some problems, specific mutation operators that profit from the features of the considered problem have been defined. They are explained in the chapters devoted to the corresponding optimisation problems.

Finally, the following crossover operators have been applied:

- One Point Crossover (OPX) [126]: It works by choosing a random gene, and then splitting both parents at this point and creating the two children by exchanging the tails. It can be used with real-coded genes as well as with binary strings.

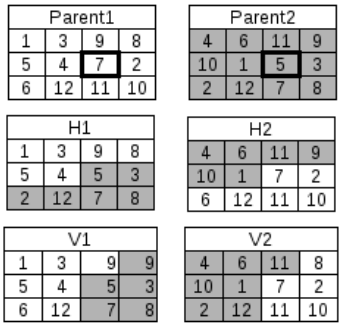


Figure 2.2: Sub-string Crossover (SSX)

- Uniform Crossover (UX) [229]: This crossover works by treating each gene independently and making a random choice as to which parent it should be inherited from. This is implemented by generating a string with L random variables from a uniform distribution over $[0, 1]$, where L is the number of genes. In each position, if the value is below a given value (usually 0.5), the gene is inherited from the first parent; otherwise, it is inherited from the second parent. The second offspring is created using the inverse mapping. It can be used with real-coded genes and with binary strings.

- Simulated Binary Crossover (SBX) [76]: This crossover simulates the operation of the OPX with binary encoding, but it was designed for real-coded genes. For doing it, a probability distribution similar to the one generated with the binary OPX is used. Such a distribution can be adapted to the problem with the parameter η_c. In every experiment $\eta_c = 5$ has been used.

- Two-Dimensional Substring Crossover (SSX) [127]: This crossover is used with two-dimensional chromosomes. First, a gene position is selected as the division point. Then, the operator randomly decides to do a vertical or horizontal operation, i.e. the chromosome is converted into a one dimensional chromosome sorting the genes by rows or by columns. Finally, the OPX operator is applied considering the converted chromosome. SSX is illustrated in Figure 2.2. $H1$ and $H2$ are generated by means of a horizontal crossover, while $V1$ and $V2$ are generated by the application of the vertical one. In both cases, the position $(3, 2)$ has been selected as the division point.

- Specific crossover: In some problems, specific crossover operators that profit from the features of the considered problem have been defined. They are explained in the chapters devoted to the corresponding optimisation problems.

The previous description is a unifying view of evolutionary approaches [93]. However, it its origin, several computer scientist independently invented different evolutionary methods [183], using different names to refer to them. Evolution Strategiess (ESs) were introduced in [207] as a method to solve optimisation problems with real-valued parameters. The idea was further developed by Schwefel [213]. One of the main contributions of them was the usage of self-adaptation. Specifically, the mutation was performed using a Gaussian distribution whose variance was coevolved with the individuals. At the same time, Genetic Algorithms (GAs) were proposed by Holland [126]. In such a case, the proposal worked on binary strings rather than on real-valued genes. Another novel contribution was the definition of the genetic operator. Other well-known schemes are Evolutionary Programming (EP) [102] which encodes the individuals as finite state machines, and Genetic Programming (GP) [148], which encodes the individuals as trees. The differences among the schemes were not based only on the encoding. For instance, in the case of the original GAs, both crossover and mutation operators were used; while in ESs the random mutation was the only source of variation. In addition, GAs used a fixed mutation probability, while in ESs the genes were mutated based on a distribution which was also evolved.
In the last years the boundaries between the different flavours of evolutionary algorithms have broken down [183]. The reason is that several evolutionary approaches that merge the ideas of them have emerged. For instance several genetic algorithms with self-adaptive mechanisms have been proposed [220]. Such proposals incorporate ideas from GAs, and from ESs, so it is not clear which is the most appropriate notation to use. For this reason, in this dissertation these algorithms are mainly referred to as EAs. However, in some parts the terms ESs and GAs are also used to refer to specific approaches that were originally published using such terms.

Eshelman's cross generational elitist selection, heterogeneous recombination, and cataclysmic mutation

The Eshelman's cross generational elitist selection, heterogeneous recombination, and cataclysmic mutation (CHC) algorithm is a specific flavour of EA that has been used in this research. CHC was proposed by Eshelman [95]. It combines a conservative selection strategy - the RW-S scheme - with a radical highly disruptive recombination operator that produces offspring that are maximally different from both parents. In addition, a method to detect convergence is incorporated on the scheme. When convergence is detected, a restart mechanism is applied. The rest of the operations follow the normal behaviour of EAs.
In CHC the mating selection is performed in such a way that individuals that are

too similar (Hamming distance below a given threshold) cannot mate each other. First, each member of the population is copied to the mating pool. Then, they are randomly paired for recombination. The recombination is made using a special procedure known as Half Uniform Crossover (HUX). This procedure first copies the common information for both parents into both children. Then, it translates half of the diverging information from each parent to each of the children. This is done with the aim of preserving the maximum amount of diversity in the population, as no new diversity is introduced during the normal iterations (mutation is not applied in the normal generations). The threshold that controls the mating selection is progressively reduced to encourage the production of new solutions when the population begins to converge. When convergence is finally reached (population not changed for a number of specified generations) a special mechanism is used to generate new diversity: the restart scheme. In such a scheme, all the solutions except the very best ones are significantly modified by applying a mutation operator with a high mutation rate.
The following components must be fixed by the practitioner:

- N: Population size.
- Cataclysmic mutation: It is usually the flip mutation operator.
- P_m: It represents the probability of the cataclysmic mutation.
- P_c: Crossover probability.
- $ConvGen$: Number of generations to detect convergence.

2.1.2 Differential Evolution

Differential Evolution (DE) is a population-based metaheuristic proposed by Storn and Price [227]. Since its inception, DE has earned a reputation as a very effective global optimiser [205]. The DE algorithm has gradually become more popular mainly because it has demonstrated good convergence properties and because of its conceptual simplicity and ease of use [226]. DE originated with the Genetic Annealing Algorithm (GAA) [204]. GAA is based on a combination of GAs and SA techniques. Specifically, it implements a set of annealing procedures that are specified via several thresholds. DE introduces two main modifications. First, candidate solutions are encoded with a set of floating-point values instead of bit-string encoding. Moreover, it incorporates the idea of using vector differences for perturbing the vector population. The idea was applied by designing a novel mutation operator which is currently known as the differential mutation operator.

Algorithm 2 Differential Evolution

```
1: Generate an initial population (P) with N individuals
2: Evaluate all individuals in the population
3: while (not stopping criterion) do
4:   for i = 1 → N do
5:     Select a random individual different from P_i (P_a)
6:     Select a random individual different from P_i and P_a (P_b)
7:     Select a random individual different from P_i, P_a and P_b (P_base)
8:     j_rand = random_integer([1, numGenes])
9:     for j = 1 → numGenes do
10:      if ((random[0, 1] <= CR) || (j == j_rand)) then
11:        Offspring_{i,j} = P_{base,j} + F * (P_{a,j} - P_{b,j})
12:      else
13:        Offspring_{i,j} = P_{i,j}
14:      end if
15:    end for
16:    NewPop_i = better(Offspring_i, P_i)
17:  end for
18:  P = NewPop
19: end while
```

Algorithm 2 shows a general pseudocode of the approach. First, an initial population with N individuals is created (line 1) and evaluated (line 2). Then, until the stopping criterion is reached (line 3) a set of steps are repeated. Such a set of steps is usually referred to as a generation of the DE scheme. The steps are the following. First, for each individual of the population (P_i) (line 4), three different individuals of the population are randomly selected (line 5-7). They are named P_a, P_b and P_{base}. Considering such individuals a new offspring is created. The differential mutation operator is used (line 8-15). The offspring is compared with P_i, and the best of both individuals is copied to a new population set (line 16). Finally, the new population set replaces the old population (line 18), and the DE generations continue.
The following parameters must be fixed:

- N: Population Size.
- CR: It represents the crossover factor. It controls the probability of choosing the mutated value instead of the current value of P_i.
- F: It Controls the amplification of differential variations.

Algorithm 3 Population-based Incremental Learning

```
1: Initialise probability vector (P)
2: while (not stopping criterion) do
3:    Generate population with N individuals according to probability vector P
4:    Evaluate all individuals in the population
5:    Find the individual with the maximum fitness in the population (max)
6:    for i = 1 → numGenes do
7:       Update P: P_i = P_i * (1 - LR) + max_i * LR
8:       if (random[0, 1] < p_m) then
9:          Mutate P : P_i = P_i * (1 − MUT_A) + random(0 or 1) * (MUT_A);
10:      end if
11:   end for
12: end while
```

2.1.3 Population-based Incremental Learning

Population-based Incremental Learning (PBIL) is a population-based metaheuristic proposed by Baluja [21]. It combines EAs with competitive learning processes typically used in artificial neural networks. PBIL abstracts away the crossover operator and redefine the role of the population. Specifically, it is an extension of the Equilibrium Genetic Algorithm (EGA) achieved through reexamination of the performance in terms of competitive learning. PBIL attempts to create a probability vector from which samples can be drawn to produce the next generation's population. Therefore, it can be seen as an EA in which a probability vector is evolved rather than individual members. The algorithm is simpler than a standard GA, but in some cases leads to better results than a standard genetic approach [22].

Algorithm 3 shows a general pseudocode of the approach. First a probability vector (P) is initialised (line 1). Each position of such a vector represents the probabilities of generating the different accepted values of a gene. In this research, PBIL has been used with binary string encoding, so each position represents the probability of generating a "one" for the corresponding gene. The initialisation assigns the value 0.5 to each position. Then, the search loop is repeated until some stopping criterion is satisfied (line 2). In each cycle, a population of N individuals is generated (line 3). The generation is performed according to the probability vector P. Then, the individuals of the population are evaluated (line 4), and the one with maximum fitness is stored in max (line 5). Afterwards, a learning procedure (line 7), and a mutation operator (line 8-10) are applied to update each position of P (line 6). The next generation considers the updated probability vector. Finally, when the stopping criterion is satisfied, the best solution found so far is returned.

The practitioner must assign a value to the following parameters:

- N: It represents the population size.
- p_m: It represents the probability of mutating each position of P.
- MUT_A: It represents the strength of the mutation.
- LR: It represents the strength of the learning procedure, or learning rate.

2.1.4 Local Search with Heuristic Restarts

Local Search with Heuristic Restart (LSHR) extends a local search algorithm by adding periodically guided perturbations to avoid local minima [165]. In short, it periodically restarts local search by means of a probability distribution F learned during the search process. It is used for combinatorial problems. F is a matrix with dimension $M \times N$, where M is the number of variables of the problem, and N the amount of valid values for each variable. The matrix is updated in a similar fashion as PBIL.
Algorithm 4 shows a general pseudocode of the approach. The probability distribution F is initialised with uniform probabilities (line 1). Then, an initial solution S is generated from F by means of a roulette-wheel based procedure (line 2). Then, the search loop is repeated until some stopping criterion is satisfied (line 3). In each cycle the following steps are executed. First, a local search is applied on current solution S (line 4). The best solution found is kept is S^* (line 5). Then, the matrix F is updated following the same rule than in PBIL. Finally, a new solution is generated using the new matrix F (line 6).

Algorithm 4 Local Search with Heuristic Restarts

```
1: initialise(probability matrix F)
2: S* ← S ← generateFrom(F)
3: while not time-limit do
4:    S ← localSearch(S)
5:    S* ← best(S,S*)
6:    F ← update(F,S*)
7:    S ← generateFrom(F)
8: end while
```

Algorithm 5 Scatter Search

```
1: Generate an initial population (P) with N individuals
2: Evaluate all individuals in the population
3: RefSet = generateReferenceSet(P)
4: while (not stopping criterion) do
5:    SubSet = SubSetGenerator(RefSet)
6:    SubSet = combinationMethod(SubSet)
7:    RefSet = generateReferenceSet(SubSet ∪ RefSet)
8: end while
```

2.1.5 Scatter Search

Scatter Search (SS) is a population-based metaheuristic that provides unifying principles for joining solutions based on generalised path constructions [111]. It utilises strategic designs where other approaches resort to randomisation [113]. SS derives its foundations from strategies originally proposed for combining decision rules and constraints in the context of integer programming [110]. It works with a small set composed of the most representative solutions from the population. Such a set is called the reference set. In order to decide which solutions are included in the reference set, both quality and diversity metrics are considered. In the used implementation, one solution is included because of its quality (the best one), while for the rest of the solutions a diversity metric is considered. Specifically, solutions are selected with the aim of maximising the Euclidean distance among them in the search space. SS has been successfully applied to several practical optimisation problems [89, 118].

Algorithm 5 shows a general pseudocode of the approach. First, an initial population with N individuals is created (line 1) and evaluated (line 2). The best $RefSize$ solutions, considering both quality and diversity, are included in a reference set (line 3). Then, until the stopping criterion is reached (line 4) a set of steps are repeated. Such a set of steps is usually referred to as a generation of the SS scheme. The steps are the following. First, a subset generation method is applied (line 5). It creates all possible subsets of size two from the reference set. The next step is to apply the solution combination method (line 6). In the used implementation, solutions are combined in a pair-wise way, so the combination method is analogous to the crossover operation. Finally, a new reference set is created by selecting the best $RefSize$ solutions, considering both the new generated solutions and the original reference set. The solution returned at the end of the execution is the best one included in the reference set.

Algorithm 6 Iterated Local Search

```
1: σ0 = generateInitialSolution();
2: σ = localSearch(σ0);
3: UpdateBestSolution(σ)
4: while (not stopping criterion) do
5:    σ' = Perturbation(σ, history)
6:    σ'' = LocalSearch(σ')
7:    σ = acceptanceCriterion(σ, σ'', history)
8:    UpdateBestSolution(σ'')
9: end while
```

2.1.6 Iterated Local Search

Iterated Local Search (ILS) [161] is a neighbourhood exploration paradigm whose essence can be given in a nut-shell: one iteratively builds a sequence of solutions generated by an embedded heuristic (usually a local search), leading to better solutions than if one were to use repeated random trials of that heuristic. The idea was initially proposed by Baxter [25] but it has been rediscovered by many authors leading to many different names like Iterated Descent [24], Large-step Markov Chain [174], or Chained Local Optimisation [173]. There are two main points that make an algorithm an iterated local search:

- There must be a single chain that is being followed.
- The intensification step occurs in a reduced space defined by the output of a black-box heuristic. Such a block box heuristic is usually a local search.

ILS is a simple and generally applicable metaheuristic which iteratively apply a local search to the current solution. ILS improves the performance of local searches by allowing them to escape local-optima and continue the search for possible better solution. The success of ILS lies in the biased sampling of this set of local optima. Algorithm 6 shows a general pseudocode of the approach. At the start of the algorithm, an initial solution (σ_0) is generated (line 1). Afterwards, a new solution (σ) is generated by applying local search (line 2), and the best solution is updated if required (line 3). Then, the search loop is repeated until some stopping criterion is satisfied (line 4). In each cycle, a diversification step is applied by perturbing σ, to obtain σ' (line 5). Intensification is then performed around σ' by applying a local search to produce a new solution σ'' (line 6). If σ'' satisfies an acceptance criterion, it replaces σ and the next cycle is carried out from this new solution (line 7). The best solution found so far is updated in each cycle if required (line 8). Finally, the best found solution is returned.

In order to obtain a particular configuration of ILS several details have to be specified. Specifically, the next components must be established:

- Method to generate the initial solution. Usually, very simple methods are used for the generation of the initial solution.

- Local Search Method. Several schemes have been tested in the literature. In this research a Hill Climbing method has always been used. During the local search, the complete neighbourhood is generated. From the obtained neighbours, the best one is selected. The process is repeated from the chosen solution. The process finishes when the local search reaches a local optimum or based on a maximum execution time.

- Perturbation strategy. ILS should lead to good biased sampling as long as the perturbations are neither too small nor too large. If they are too small, one will often fall back to the same local optimum. If on the contrary the perturbations are too large, the generated solution will be random, so a random restart type algorithm is obtained. Since deterministic perturbations may lead to short cycles, randomised perturbations have been used. In addition, the perturbation scheme might depend on the history of the execution. In the schemes developed in this research the strength of the perturbation method is increased when stagnation is detected. The perturbation strategies are detailed in the chapters devoted to the practical optimisation problems.

- Acceptance criterion. The acceptance criterion is in charge of picking which solution between σ and σ'' is used in the next cycle. In this research, a fixed rule that involves selecting the best one has always been applied. Most of the work using ILS has been of this type [161]. In other works, such a decision also considers the history of the execution.

2.1.7 Variable Neighbourhood Search

Variable Neighbourhood Search (VNS) is a modern metaheuristic introduced by Mladenović and Hansen [184]. It has been successfully applied to several practical optimisation problems [33, 168]. Its basic idea is to systematically change the neighbourhood structure with the aim of avoiding stagnation. VNS is based on three simple facts [111]:

- A local minimum with respect to one neighbourhood structure is not necessary so with another.

Algorithm 7 Variable Neighbourhood Search

1: σ = generateInitialSolution();
2: **while** (not stopping criterion) **do**
3: Generate random solution $\sigma' \in N_k^s(\sigma)$
4: $\sigma'' = \text{Local\_Search}(N_l^{LS}(\sigma'))$
5: **if** $(f(\sigma'') < f(\sigma))$ **then**
6: $\sigma = \sigma''$;
7: **end if**
8: UpdateBestSolution(σ'')
9: **end while**

- A global minimum is a local minimum with respect to all possible neighbourhood structures.
- For many problems local minima with respect to one or several neighbourhoods are relatively close to each other.

The core of the VNS is based on using two kind of schemes: shake and local search. Intensification is achieved by the local search while the shaking of the neighbourhood structure acts as a diversification mechanism. In order to develop an effective VNS algorithm, two kinds of neighbourhood structures are required: $N_k^s(x)$ and $N_l^{LS}(x)$. The first one is used for shaking, while the second one is used for local search. Multiple neighbourhood of each kind might be used. For this reason, a subscript has been added in the notation of the neighbourhoods.
Algorithm 7 shows a general pseudocode of the approach. First, an initial solution is generated (line 1). Then, until the stopping criterion is satisfied (line 2), a set of iterations are repeated. In each iteration the following steps are executed. First, the shaking procedure is executed (line 3). It lies in selecting a random solution from N_k^s. The way in which k is selected depends on the implementation, but usually, the value of k is initialised to one, and incremented on each cycle. Then, a local search procedure is executed (line 4). It also depends on the implementation. Usually a Variable Neighbourhood Descent (VND) is used. In such a case, every $N_l^{LS}(x)$ is checked in a deterministic way. If the new generated solution improves the current one, such a solution is updated (lines 5-7). Finally, the best found solution is updated if required (line 8).
The components which must be fixed by the practitioner are the following:

- Method to generate the initial solution.
- The set of neighbourhoods for shaking: N_k^s.

- The set of neighbourhoods for intensification: N_l^{LS}.
- The local search procedure.

2.1.8 Simulated Annealing

Simulated Annealing (SA) is a trajectory-based optimisation technique used to address discrete and, to a lesser extent, continuous optimisation problems. It was independently proposed by Kirkpatrick *et al.* [141] and by Cerný [238]. Subsequently, is has been successfully applied to many optimisation problems [2, 115]. SA is so named because of its analogy to the process of physical annealing with solids [111]. In such a process a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration. The heat causes the atoms to become unstuck from their initial positions and wander randomly through states of higher energy. The slow cooling gives them more chances of finding configurations with lower internal energy.
SA establishes the connection between this type of thermodynamic behaviour and the search for global optima. Specifically, it makes use of the Boltzmann distribution from statistical mechanics, i.e. the distribution of energies in an idealised many particle system. One of the key features of SA is that it provides a mean to escape local optima by allowing moves which worsen the objective function value. Such moves are accepted with a probability that depends on a parameter of the approach which represents the temperature. The probability of accepting such moves is inspired on the Boltzmann distribution.
Algorithm 8 shows a general pseudocode of the approach. The algorithm works iteratively and keeps a single tentative solution at any time. First, an initial solution is generated (line 1) and stored as the best solution (line 2). Usually, very simple methods are used for the generation of the initial solution. A parameter that represents the temperature (t) is initialised (line 3). Usually, a high value is used. Then, until the stopping criterion is reached (line 4) a set of iterations are repeated (line 6) with a fixed temperature value. By analogy with the thermodynamic process, each iteration of the SA attempts to replace the current solution by a new generated solution. Such solution (σ_n) is picked randomly from the neighbourhood of σ (line 7). The best solution found so far is updated if required (line 8). The generated solution might replace σ depending on an acceptance criterion (lines 9-14). Such an acceptance criterion depends on the current temperature and on the fitness of the generated solution. When the iterations have been completed, the temperature that controls the Boltzmann distribution is decreased (line 16). Such a step is usually named the cooling process. Several cooling processes have been defined [228].

Algorithm 8 Simulated Annealing

```
1: σ = generateInitialSolution();
2: UpdateBestSolution(σ)
3: t = startingTemperature;
4: while (not stopping criterion) do
5:    while ((notStoppingcriterion) and (t > temperatureLimit)) do
6:       for i = 1 → maxIterations do
7:          Generate a solution σ_n ∈ N(σ)
8:          UpdateBestSolution(σ_n)
9:          Δ := f(σ) − f(σ_n);
10:         if Δ ≥ 0 then
11:            σ = σ_n;
12:         else if random([0,1]) ≤ e^(Δ/t) then
13:            σ = σ_n;
14:         end if
15:      end for
16:      t = α ∗ t;
17:   end while
18:   t = startingTemperature;
19:   σ = σ_b;
20: end while
```

In this research, a static cooling process has been considered. The same steps are repeated until the temperature is lower than a threshold (line 5). When the temperature is lower than the threshold, it is reset to its initial value (line 18), and the local search continues starting from the best known solution (line 19). When the stopping criterion is reached, the best solution found so far is returned.
The practitioner must assign a value to the next parameters:

- MaxIterations: It represents the number of iterations performed with a fixed temperature value.
- StartingTemperature: It represents the initial temperature, and the value to which the temperature is reset when is lower than a specified limit.
- TemperatureLimit: It represents the threshold for the minimum accepted temperature. When the temperature is lower than this value, it is reset.
- α: It represents the cooling factor.

Algorithm 9 Greedy Randomised Adaptive Search Procedure

```
1: while (not stopping criterion) do
2:    σ = Greedy_Randomised_Construction()
3:    σ' = Local_Search(σ)
4:    UpdateBestSolution(σ')
5: end while
```

2.1.9 Greedy Randomised Adaptive Search Procedure

Greedy Randomised Adaptive Search Procedure (GRASP) is a very popular multi-start metaheuristic especially suited for combinatorial optimisation problems [111]. It was proposed by Feo and Resende [97]. GRASP typically consists of iterations made up from successive constructions of a greedy randomised solution, and subsequent iterative improvement. The iterative improvement is usually performed with a local search strategy. It has been successfully applied to several practical optimisation problems [27, 155].

Algorithm 9 shows a general pseudocode of the approach. A set of iterations are repeated until the stopping criterion is satisfied (line 1). The iterations consist of two steps. First, a greedy randomised construction procedure is executed (line 2). Such a phase builds a feasible solution with a stochastic approach. The construction is performed in the following way. First, an empty solution is considered. Then, at each step, a set with all elements that can be incorporated to the partial solution under construction without destroying feasibility is calculated. The elements of the set are sorted considering a greedy function that usually represents the increase in the fitness function due to the incorporation of each element to the partial solution. The best elements are inserted in a restricted candidate list (RCL). Finally, an element of RCL - randomly selected - is incorporated to the partial solution. The elements of RCL are selected with the same probabilities independently of their involved fitness increase. The process finishes when no elements can be inserted without destroying the feasibility. The solutions returned by the construction phase are not guaranteed to be locally optimal. For this reason, such a solution undergoes a second stage, based on applying a local search procedure (line 3). Finally, considering the new generated solution, the best solution is updated if required (line 4).

The components which must be fixed by the practitioner are the following:

- Local search procedure.
- Size of RCL ($|RCL|$).

2.2 Multi-Objective Metaheuristics

2.2.1 Multi-Objective Evolutionary Algorithms

The application of EAs in multi-objective optimisation has received a growing interest in the last decades [74]. Such approaches are named Multi-Objective Evolutionary Algorithms (MOEAs). They follow the same scheme that the presented for the mono-objective evolutionary approaches (Algorithm 1). However, the different components are designed with the aim of obtaining an approximation of the Pareto Front. Most research in this area has concentrated on the selection stage due to the need to integrate vectorial performance measures in such a phase. As in the case of mono-objective EAs, several different schemes have been proposed [108].
Early applications to Multi-Objective Optimisation Problems (MOPs) were mainly preference-based approaches, i.e. the objectives were weighted and mono-objective EAs were applied [74]. Vector Evaluated Genetic Algorithm (VEGA) was the first EA capable of finding multiple trade-off solutions in a single run. Therefore, it can be considered as the first MOEA. It was proposed by Schaffer [212]. Goldberg suggested a sketch for MOEAs based on the concept of domination [114]. Since then, several approaches have been proposed:

- Non-Dominated Sorting Genetic Algorithm (NSGA) [224], proposed by Srinivas and Deb.
- Niched Pareto Genetic Algorithm (NPGA) [129], proposed by Horn *et al.*.
- Multi-Objective Optimisation Genetic Algorithm (MOGA) [103], proposed by Fonseca and Flemming.
- The weighted-sum approach [133], proposed by Ishibuchi and Murata.
- Strength Pareto Evolutionary Algorithm (SPEA) [258], proposed by Zitzler and Thiele.
- Pareto Archived Evolution Strategy (PAES) [145], proposed by Knowles and Corne.
- Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [78], proposed by Deb *et al.*
- Strength Pareto Evolutionary Algorithm 2 (SPEA2) [257] proposed by Zitzler *et al.*

- Indicator-Based Evolutionary Algorithm (IBEA) [256], proposed by Zitzler and Künzli.
- Multi-Objective Cellular Genetic Algorithm (MOCell) [189], proposed by Nebro *et al.*

Among the previous schemes the next ones have been used in this research: NSGA-II, SPEA2, IBEA, and MOCell. In the following sections, a description of each one of them is presented.

Non-Dominated Sorting Genetic Algorithm II

NSGA-II [78] is a non-dominated sorting based MOEA. Two of the most important characteristics of this algorithm are the following. First, it uses a fast non-dominated sorting approach. Second, it applies a selection operator which combines previous populations with new generated ones, ensuring elitism in the approach. Both aforementioned features are guided by the *crowded comparison operator* ($\geq_n$). This operator assigns two attributes to every individual i of the population: the *non-domination rank* (i_{rank}) and the *local crowding distance* ($i_{distance}$). The non-domination rank makes use of the Pareto Dominance concept. The procedure to calculate it is as follows. First, the set of non-dominated individuals of the population are assigned to the first rank. Then, the process is repeated considering only the individuals that do not have a rank assigned. The rank assigned at each step is increased by one. The process ends when all individuals in the population have their corresponding rank established.
The local crowding distance is used to estimate the density of solutions surrounding a particular individual. First, the size of the largest cuboid enclosing the individual i without including any other individual that belongs to its rank is calculated. Then, the crowding distance is calculated as the mean side-length of the cuboid. It is worthy to mention that the local crowding distance of the boundary individuals of every rank is assigned to an infinite value.
Finally, the partial order given by $\geq_n$ is the following:

$$i \geq_n j \quad if \quad ((i_{rank} < j_{rank}) \; or \; ((i_{rank} = j_{rank}) \; and \; (i_{distance} > j_{distance}))) \tag{2.1}$$

Algorithm 10 shows the pseudocode of the approach. First, an initial population with N individuals is created (line 1) and evaluated (line 2). Then, until the stopping criterion is reached (line 3) a set of steps are repeated. The steps are the following. First, the fitness of every individual in the population is calculated (line 4). In the

Algorithm 10 Non-Dominated Sorting Genetic Algorithm II

1: **Initialisation**: Generate an initial population (P) with N individuals
2: **Evaluation**: Evaluate all individuals in the population
3: **while** (not stopping criterion) **do**
4: **Fitness assignment**: Calculate fitness values of individuals in P_t. Use only the non-domination rank in the first generation, and the crowded comparison operator in other generations.
5: **Mating selection**: Perform binary tournament selection on P_t in order to fill the mating pool.
6: **Variation**: Apply genetic operators to the mating pool to create a child population CP.
7: **Evaluation**: evaluate individuals in CP.
8: **Survivor selection**: Combine P and CP, selecting the best individuals using the crowded comparison operator to constitute the new P.
9: **end while**

first generation it calculates the rank of each individual. In the following generations both the rank, and the crowding distances are calculated. Then, the mating pool is created, i.e. a set of N parents are selected (line 5). They are selected by means of binary tournaments. The tournaments consider the fitness calculated in the previous step. Then, the variation stage is applied (line 6). The crossover and mutation operators are used to generate the child population (CP). Such child population is evaluated (line 7). Finally, the survivor selection scheme (line 8) lies in selecting the best N individuals between P and CP. The crowded comparison operator is used to perform the comparisons. The next generation considers the new generated population.
The following components must be assigned by the practitioner:

- N: It represents the population size.
- Mutation operator and mutation probability (p_m).
- Crossover operator and crossover probability (p_c).

Strength Pareto Evolutionary Algorithm 2

SPEA2 is a well-known MOEA proposed by Zitzler *et al.* [257]. It incorporates the usage of an external archive to manage the best found solutions. SPEA2 establishes an order among the individuals using a fine-grained fitness assignment strategy. Such strategy assigns to each individual a fitness value that is the sum of its strength raw

Algorithm 11 Strength Pareto Evolutionary Algorithm 2

1: **Initialisation**: Generate an initial population (P) with N individuals
2: **Initialisation**: Create an empty archive $\overline{P}$.
3: **while** (not stopping criterion) **do**
4: **Evaluation**: Evaluate all individuals in the population.
5: **Fitness assignment**: Calculate the fitness values of individuals in P and $\overline{P}$. For each individual i, calculate the raw fitness i_{raw}, and the density estimation $i_{density}$.
6: **Environmental Selection**: Create a new archive ($\overline{NP}$) with the non-dominated individuals in P and $\overline{P}$. If $|\overline{NP}| > \overline{N}$ reduce $\overline{NP}$. Otherwise, if $|\overline{NP}| < \overline{N}$, fill $\overline{NP}$ with dominated individuals in P and $\overline{P}$, considering their fitness. Establish $\overline{NP}$ as the current archive.
7: **Mating selection**: Perform binary tournament selection on $\overline{P}$ to fill the mating pool.
8: **Variation**: Apply genetic operators to the mating pool and set P to the resulting population.
9: **end while**

fitness plus a density estimation. The density information ($i_{density}$) is incorporated to discriminate among individuals which have identical raw fitness values. In order to calculate the raw fitness, the strength $i_{strength}$ of each individual i is calculated as the number of solutions that it dominates, considering the population (P) and the archive ($\overline{P}$):

$$i_{strength} = |\{j | j \in P \cup \overline{P} \wedge i \preceq j\}| \tag{2.2}$$

Then, the raw fitness i_{raw} is calculated as follows:

$$i_{raw} = \sum_{j \in P \cup \overline{P}, j \preceq i} j_{strength} \tag{2.3}$$

Algorithm 11 shows the pseudocode of the approach. First, an initial population with N individuals (line 1) and an empty archive (line 2) are created. Then, until the stopping criterion is reached (line 3) a set of steps are repeated. The steps are the following. First, the current population is evaluated (line 4). Then, the fitness assignment strategy is used to calculate the fitness of the individuals in the population and archive (line 5). In each generation the non-dominated individuals of both the population and the archive are used to update the archive; if the number of non-dominated individuals is greater than the desired archive size, a truncation operator based on calculating the distances to the k-th nearest neighbour is used. Otherwise,

if the number of non-dominated solutions is lower than the desired archive size, the best dominated individuals of the old archive or population are used to fill the archive. All this procedure is known as *Environmental Selection* (line 6). Then, the mating pool is created (line 7). The N parents are selected by means of binary tournaments. Finally, the variation stage is applied (line 8) to generate the new population. The next generation considers the new generated population.
The following components must be assigned by the practitioner:

- N: It represents the population size.
- $\overline{N}$: It represents the desired archive size.
- Mutation operator and mutation probability (p_m).
- Crossover operator and crossover probability (p_c).

Indicator-based Evolutionary Algorithm

IBEA [256] is a MOEA proposed by Zitzler and Künzli that allows defining the optimisation goal in terms of a *binary quality indicator*. This measure is used directly for fitness calculation. IBEA allows the usage of different binary quality indicators. In this thesis the binary multiplicative ϵ-indicator [259] has been used. There exist two versions of IBEA, the basic one and the adaptive version. In the adaptive version, objectives values are normalised, and the indicator values are adaptively scaled. The basic approach is similar, but no scaling or normalisation is performed. Both versions of the algorithm have been used.
Algorithm 12 shows the pseudocode of the adaptive version. First, an initial population with N individuals is created (line 1) and evaluated (line 2). Then, until the stopping criterion is reached (line 3) a set of steps are repeated. First, the fitness of individuals in the population is calculated (line 4). The objective values must be normalised and the indicator values calculated (lines 4.1). The formula in line 4.2 is used to calculate the fitness value of each individual. The fitness function depends on a scaling factor k. Then, the environmental selection is performed (line 5). During the environmental selection the worst individual, i.e. the individual with lowest fitness is removed. This step is repeated until the population size does not exceed N. Each time an individual is removed, the fitness of the remaining individuals is recalculated. The mating pool is constituted by performing a binary tournament selection with replacement over the current population (line 6). Finally, a variation process over the individuals in the mating pool is performed (line 7) and the new individuals are attached to the current population.

Algorithm 12 Indicator-based Evolutionary Algorithm (Adaptive Version)

1: **Initialisation**: Generate an initial population (P) with N individuals
2: **Evaluation**: Evaluate all individuals in the population
3: **while** (not stopping criterion) **do**
4: **Fitness assignment**: calculate the fitness values using the quality indicator.

1. Calculate indicator values $I(x^1, x^2)$ using the normalised objective values f_i' and determine the maximum absolute value $c = max_{x^1,x^2 \in P}|I(x^1, x^2)|$.
2. $\forall x^1 \in P\,, F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-I(\{x^2\},\{x^1\})/(c \cdot k)}$.

5: **Environmental selection**: until the size of P does not exceed N, remove the individual with the smallest fitness value, and recalculate the fitness value of the remaining individuals.
6: **Mating selection**: Perform binary tournament selection with replacement on P in order to fill the temporary mating pool P'.
7: **Variation**: Apply recombination and mutation operators to the mating pool P' and add the resulting offspring to P.
8: **end while**

The following components must be assigned by the practitioner:

- N: It represents the population size.
- k: It represents the fitness scaling factor.
- Mutation operator and mutation probability (p_m).
- Crossover operator and crossover probability (p_c).

Multi-objective Cellular Genetic Algorithm

MOCell is a Cellular Genetic Algorithm (CGA) proposed by Nebro *et al.* [189]. In CGAs, each member of the population is assigned to a point on a grid. Recombination and selection considers solely the neighbours of a given individual. In addition, the considered version includes an external archive to store the non-dominated solutions found so far. This archive is bounded and uses the crowding operator of NSGA-II to keep the diversity of the inserted solutions.
The asynchronous version named aMOCell4 [190] has been used. Algorithm 13 shows the pseudocode of the approach. First, an initial population with N individuals is created (line 1) and evaluated (line 2). The non-dominated individuals

Algorithm 13 Multi-objective Cellular Genetic Algorithm

```
1: Initialisation: Generate an initial population (P) with N individuals
2: Evaluation: Evaluate all individuals in the population
3: updateArchive(P̄)
4: while (not stopping criterion) do
5:    for individual ← 1 to N do
6:       neighbours ←getNeighborhood(population, position(individual))
7:       neighbours.add(position(individual))
8:       parent1 ←selection(neighbours)
9:       parent2 ←selection(archive)
10:      offspring←recombination(Pc, parent1, parent2)
11:      offspring←mutation(Pm, offspring)
12:      evaluate(offspring)
13:      evaluateFitness(offspring)
14:      updateArchive(offspring)
15:      replacement(position(individual), offspring)
16:   end for
17: end while
```

are included in the archive (line 3). Then, until the stopping criterion is reached (line 4) a set of steps are repeated. Specifically, for each member of the population (line 5), the following steps are executed. First, the neighbours of the individual are calculated (line 6). The individual itself is included in the list of neighbours (line 7). Then, two parents are selected. The first one is chosen from the neighbourhood (line 8), while the second one is chosen from the archive (line 9). The selection is performed with a binary tournament. An offspring is created by using recombination (line 10), and mutation (line 11). Then, the offspring is evaluated (line 12), and its fitness is calculated considering the NSGA-II crowding operator (line 13). The new individual is used to update the archive (line 14). Finally, the replacement scheme is executed. The new offspring is compared with the current one, replacing it if better. In the case of both solution be non-dominated, the worst individual in the neighbourhood (considering the crowding operator) is replaced by the current one. The next generation continues with the new generated population.
The following components must be assigned by the practitioner:

- N: It represents the population size.
- Mutation operator and mutation probability (p_m).
- Crossover operator and crossover probability (p_c).

Algorithm 14 Evolution Strategy with NSGA-II

1: Initialise population P of μ individuals
2: Evaluate all individuals in the population
3: Initialise variance to σ_0 for each individual $I \in P$
4: **while** (not stopping criterion) **do**
5: $P' = \emptyset$
6: **for** each $I = (x_1, ..., x_n, \sigma) \in P$ **do**
7: $\sigma' = \sigma e^{N(0,\Delta)}$
8: Create $I' = (N(x_1, \sigma'), N(x_2, \sigma'), ..., N(x_n, \sigma'), \sigma')$
9: Evaluate I'
10: $P' = P' \cup \{I'\}$
11: **end for**
12: $P = P \cup P'$
13: Calculate front F_1 as Non-dominated individuals of P
14: **for** $i = 2$ to n **do**
15: Generate fronts F_i as Non-dominated individuals of $P \setminus (F_1 \cup ... \cup F_{i-1})$
16: **end for**
17: Sort solutions in each F_i $(i = 1, \ldots, n)$ using the crowding distance
18: Delete the worst μ individuals in population P
19: **end while**

Evolution Strategy with NSGA-II

The Evolution Strategy with NSGA-II (ESN) algorithm is based on the hybridisation of Evolution Strategies and NSGA-II. The algorithm uses the standard Evolution Strategies' steps [15], replacing the selection process by the NSGA-II selection scheme. The mutation process implemented was the standard $(\mu + \lambda)$ process [16], although in this research every configuration has used $\lambda = \mu$.
Algorithm 14 shows the pseudocode of the approach. First, a population with μ individuals is created (line 1), and evaluated (line 2). The variance that controls the mutation operator is initialised in each individual (line 3). Then, until the stopping criterion is reached (line 4) a set of steps are repeated. First, an empty set that represents the new offspring set is created (line 5). Then, each individual of the population (line 6) undergoes the mutation step. In the mutation step, both the variance value (line 7), and the rest of the genes (line 8) are mutated. The new genes are created by using a random value from a Gaussian distribution centred in the current value, and with the corresponding variance. The new offspring is evaluated (line 9), and inserted in the offspring set (line 10). Finally, the population and offspring are joined (line 12), and the selection scheme of NSGA-II is executed

(lines 13-18). The process continues with the new generated population.
The following components must be assigned by the practitioner:

- μ: It represents the population size.
- σ_0: Initial value of the variance.

2.2.2 Multi-Objective Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is a population-based metaheuristic proposed by Kennedy and Eberhart [140]. In this kind of algorithms the members of the population are dubbed particles. It was first intended for simulating social behaviour as a representation of the movement of organisms in a bird flock or fish school. Many philosophical aspects of PSO and swarm intelligence have been studied [87].
Several multi-objective versions of PSO have been proposed. In this research the proposal of Coello *et al.* [59] has been used. Such a version combines PSO with the archiving strategy of PAES [145]. Algorithm 15 shows a pseudocode of the approach. First, an initial swarm with N particles is created (line 1) and evaluated (line 2). The velocities are initialised to zero in each dimension (line 3), and the repository is populated with the non-dominated particles (line 4). Each particle is stored as the best solution found so far in the corresponding location (lines 5-7). Then, until the stopping criterion is reached (line 8) a set of steps are repeated. Specifically, for each member of the swarm (line 9), the following steps are executed. First, a leader particle is selected from the repository (line 10). In order to select the leader, the density calculation from PAES is used. Specifically, a roulette-wheel selection is applied to favour particles in less populated regions. Then, the velocity (line 11) and position (line 12) of the particle are updated using the standard equations of PSO but using the leader particle selected from the repository, instead of best neighbour as it is usual. The new generated particle is evaluated (line 13). Then, the repository is updated with the new particle. The updating mechanism of PAES is used. Finally, the particle best position is updated if required. Specifically, if the new particle dominates the current contents of best position, it is updated. In addition, in the cases where the new particle and the contents of best position are not comparable (no dominations between them), one of them is randomly selected to keep stored as the best one.
The following components must be assigned by the practitioner:

- N: It represents the population size.
- Div: It represents the number of divisions performed in the search space to calculate the density information.

Algorithm 15 Multi-Objective Particle Swarm Optimisation

1: **Initialise the swarm** (S) with N particles
2: **Evaluate** all individuals in the swarm
3: **Initialise the velocity** of each particle to 0
4: **Update repository** with non-dominated particles
5: **for** $i = 1$ to N **do**
6: Store objectives and position of S_i as $PBestObjetives_i$ and $PBestPosition_i$
7: **end for**
8: **while** (not stopping criterion) **do**
9: **for** $= 1$ to N **do**
10: **Select Leader** (L) from the repository
11: **Update velocity** of P_i using the PSO standard equations and L as the selected neighbour.
12: **Update NewPosition** using the PSO standard equations
13: **Evaluate** NewPosition
14: **Update Repository** with NewPosition
15: Update $PBestObjective_i$ and $PBestPosition_i$
16: **end for**
17: **end while**

2.2.3 Non-dominated Sorting Differential Evolution

Several methods have been proposed to adapt the DE scheme to multi-objective optimisation. In this research, the Non-dominated Sorting Differential Evolution for Multiobjective Optimisation (NSDEMO) proposed by Iorio and Li [131] has been used. NSDEMO is a multi-objective differential evolution based on NSGA-II. The algorithm behaves as the NSGA-II approach, but it replaces the mating selection and variation stage of the NSGA-II with the one of the DE scheme. In particular, the classic differential mutation operator is used.

Algorithm 16 shows the pseudocode of the approach. First, an initial population with N individuals is created (line 1) and evaluated (line 2). Then, until the stopping criterion is reached (line 3) a set of steps are repeated. The steps are the following. First, the fitness of every individual in the population is calculated using the NSGA-II crowding operator (line 4). Then, the offspring is created (line 5-17) by using the differential evolution mutation operator. The offspring is evaluated (line 18). Finally, the survivor selection scheme (line 19) is executed. The best N individuals between the current population and the offspring are selected. The crowded comparison operator is used to perform the comparisons. The next generation considers the new generated population.

Algorithm 16 Non-dominated Sorting Differential Evolution

1: **Initialisation**: Generate an initial population (P) with N individuals
2: **Evaluation**: Evaluate all individuals in the population
3: **while** (not stopping criterion) **do**
4: **Fitness assignment**: Calculate fitness values of individuals in P_t using the crowding operator.
Offspring creation:
5: **for** $i = 1 \rightarrow N$ **do**
6: Select a random individual different from P_i (P_a)
7: Select a random individual different from P_i and P_a (P_b)
8: Select a random individual different from P_i, P_a and P_b (P_{base})
9: $j_{rand} = random\_integer([1, numGenes])$
10: **for** $j = 1 \rightarrow numGenes$ **do**
11: **if** $((random[0,1] <= CR)$ || $(j == j_{rand}))$ **then**
12: $CP_{i,j} = P_{base,j}$ + F * ($P_{a,j}$ - $P_{b,j}$)
13: **else**
14: $CP_{i,j} = P_{i,j}$
15: **end if**
16: **end for**
17: **end for**
18: **Evaluation**: evaluate individuals in CP.
19: **Survivor selection**: Combine P and CP, selecting the best individuals using the crowded comparison operator to constitute the new P.
20: **end while**

The following parameters must be fixed:

- N: Population Size.
- CR: It represents the crossover factor. It controls the probability of choosing the mutated value instead of the current value of P_i.
- F: It Controls the amplification of differential variations.

2.3 Memetic Algorithms

Memetic Algorithms (MAs) are population-based metaheuristics composed of an evolutionary framework and a set of local search procedures which are activated

Algorithm 17 Memetic Algorithm

1: **Initialisation**: Generate an initial population with N individuals
2: **Evaluation**: Evaluate all individuals in the population
3: **while** (not stopping criterion) **do**
4: **Mating selection**: select parents to generate the offspring
5: **Variation**: Apply genetic operators to the mating pool to create a child population
6: **Evaluation**: Evaluate the child population
7: **Individual learning**: Perform individual learning process in the population with a probability p_l
8: **Survivor selection**: Select individuals for the next generation
9: **end while**

within the generation cycle of the external framework [191]. Its definition was proposed by Moscato [186]. The first implementation was given in [197]. They are of great value because they perform some orders of magnitude faster than traditional genetic algorithms in some problem domains [106]. These algorithms have been applied both for mono-objective [194] and multi-objective problems [218]. In this research, several memetic algorithms have been used. They have combined the population-based metaheuristics previously presented in this chapter with simple local search algorithms based on hill climbing.

Algorithm 17 shows the scheme of a basic MA for an EA. First, an initial population is created (line 1), and evaluated (line 2). Then, until the stopping criterion is reached (line 3) a set of steps are repeated. Such steps are similar to a generation of an EA. However, an individual learning stage (line 7) has been added. The learning step is usually applied on each generation with a probability p_l. In [132] the effect of the probability p_l was analysed. Authors concluded that the performance of MAs can be improved by dynamically changing the probability p_l. In other cases [106] the learning process has been performed in every generation. For the problems considered in this research, the initial experiments showed that using the local search in every generation was preferable. The reason is that in the considered approaches the individuals obtained after the variation phase may not have high enough quality components. In addition, they could be easily improved by the individual learning process. Thus, the best results were obtained by applying the learning process in each generation. Therefore, our approaches have considered the configuration $p_l = 1$ in every case. The rest of the memetic algorithms used in this research has been similar to it, i.e., the learning process has been executed in every generation after the variation stage of the considered population-based metaheuristics.

MAs which are based on EAs are also referred to in the literature as *Baldwinian Evolutionary Algorithms* and *Lamarckian Evolutionary Algorithms*. They differ in the way in which the individual learning is incorporated into the approach [248]. On the one hand, the *Lamarckian learning* forces the genotype to reflect the result of improvement in the learning by placing the locally improved individual back into the population to compete for reproduction. On the other hand, the *Baldwinian learning* modifies the fitness of the individuals to reflect the improvement after the learning process. However, the improved genotype is not encoded back into the population. Both kinds of approaches have been successfully applied with different optimisation problems [153]. Such kinds of learning processes can be generalised to other metaheuristics. In this research, Lamarckian learning has always been used, i.e. the solution (individuals, particles, etc.) are changed to reflect the improvement after the learning process.

Attending to other features, several classifications of MAs have been proposed. Considering the classification exposed in [193], MAs can be categorised in several generations. The first generation of MAs refers to basic hybrid algorithms that combine a population-based global search with an individual learning process. The MAs used in this research belong to this generation. The second generation of MAs include a particular case of hyperheuristic among other approaches. In this case, different local search procedures are simultaneously considered. They will compete based on their past merits in generating local improvements through a reward mechanism. The algorithm decides which procedure is selected to proceed for future local refinements. In this research novel hyperheuristics have been proposed. However, they have not been used to select among different local search schemes, but to select among different parameters and metaheuristics. Finally, in third generation MAs, in contrast to second generation MAs, the pool of memes is dynamically generated during the optimisation process, instead of specifying it a priori. They have not been considered in this work.

2.4 Parallel Metaheuristics

Solving practical optimisation problems typically involves highly constrained design optimisation tasks with high computational costs. Metaheuristics often offer the only practical approach to solving complex problems of realistic dimensions. However, the limits of what may be solved in reduced computing times are still reached rapidly for many problem settings [68]. Therefore, the high computing time of metaheuristics constitutes a major handicap for their expansion.

The desire to reduce the execution time naturally leads to considering the use of

parallel and distributed processing techniques. Moreover, the performance of metaheuristics often depends on the particular problem setting and instance characteristics. Consequently, a major issue in metaheuristics design is not only how to build them for maximum performance, but also how to make them robust. Parallel metaheuristics aim to address both issues. Thus, parallel approaches aims not only to achieve time saving by distributing the computational effort but also to get benefit from the algorithmic aspects by the cooperation between different schemes [240]. This way, a parallel metaheuristic seeks to find as good or better solutions in less time as a serial implementation using less resources and/or searching more of the solution space in the same amount of execution time, i.e. increased efficiency and effectiveness.
Several parallel metaheuristics have been presented in the literature. Crainic and Toulouse presented recently a state of the art survey of parallel metaheuristics [68]. They classified parallelisation strategies according to the source of parallelism:

- *Low-level parallelism*: The source of parallelism is found within an iteration of the metaheuristic. The implementation depends on the considered metaheuristic. The aim is to speedup computations, without any attempt at achieving a better exploration or higher quality solutions.

- *Decomposition*: Parallelism comes from the decomposition of the decision variables into disjoint subsets. The particular metaheuristic is applied to each subset. The set of visited solutions using this parallel implementation is different from that of the sequential implementation, so the comparisons methods should consider the invested times, and obtained solutions simultaneously.

- *Concurrent strategies*: Several concurrent searches are performed simultaneously. Each concurrent thread may or may not execute the same metaheuristic. In addition, they may communicate during the search or only at the end to identify the best overall solution. The latter are known as independent search methods, while the former are often called cooperative multi-thread strategies. As in the previous case, the set of visited solutions using this parallel implementation is different from that of the sequential implementation.

Several taxonomies for the parallelisation of specific metaheuristics have also been presented. They do not differ significantly from the previous taxonomy. Cantù-Paz proposed a classification for parallel GAs [48]. Three main types of parallel GAs are distinguished. The global single-population master slave GA is identical to the low-level parallelism previously presented. Two other categories classify the parallel GAs according to the size of the populations that evolve in parallel, and to the

degree of communication. They are called coarse-grained and fine-grained parallelisation strategies. Considering the proposed implementations, both of them can be considered as concurrent strategies. However, some extension of them [58] might be considered as hybrid approaches that combine parallelisation by decomposition and parallelisation by concurrent strategies.
Another taxonomy for parallel EAs was proposed by Coello. Three major parallel computational paradigms were identified [58]: *master-slave*, *diffusion*, and *island*. They map directly to the taxonomy of Cantù-Paz. In the *master-slave model*, objective function evaluations are distributed among the slave processors while a master processor executes the evolutionary operators. The master-slave model is similar to the global single-population master slave GA. The *diffusion model* deals with one conceptual population, except each processor holds only one to a few individuals. The evolutionary operators occur only within neighbourhoods, which depends on the model topological structure and design. The diffusion approach is similar to the fine-grained parallelisation strategy. Finally, the *island-based models* conceptually divide the overall population into a number of independent and separate subpopulations, i.e. there are small, separate, and simultaneously executing EAs (one per processor or island). Each island evolves in isolation for the majority of the execution, but occasionally, some individuals can be migrated between neighbour islands. The island model is the coarse-grained model.

2.4.1 Island-based Model

In this research, the proposed parallel strategy starts from the island model. As it has been mentioned, this model receives different names depending on the strategies that are applied in the islands. For instance, in PSO the name multi-swarm technique is usually utilised. In the case of GAs, multi-deme is often used. In this research the name island-based model is used independently of the strategies applied on the islands.
The island-based model, when compared to the other parallel proposals, brings two main benefits: it maps easily onto the parallel architectures (thanks to its distributed and coarse-grained structure), and it extends the search area (due to multiplicity of islands) so it might prevent from sticking in local optima. Coello analysed the kind of island-based models that had been proposed for multi-objective optimisation with EAs [58]. The analysed approaches can also be used with mono-objective optimisation. Moreover, they are not limited to EAs. Coello identified four basic island-based schemes: all islands execute identical metaheuristics and parameters (homogeneous), all islands execute different metaheuristics and/or parameters (heterogeneous), each island evaluates different objective function subsets, and each

island represents a different region of the genotype or phenotype domains. The first two variants are usually known as *standard island models*. In them, each island's population represents solutions to the same problem but are evolving in isolation. In the third variant, each island focuses on some of the considered objectives. It is used only for MOPs. The last variant isolates each processor to solve specific, non-overlapping regions of genotype/phenotype domain space. In the last variant, each island probably generates values outside its constrained region: such values can be deleted from the population or they can be sent to the appropriate island. In every case, each island's population is evolved in isolation. However, the islands exchange individuals occasionally. This exchange of individual is called migration.
The standard island models, where each processor identifies the complete search space, appears a more easy-to-implement and efficient method [240], although neither case guarantees optimality. In general, the simplicity for the implementation of standard island-based schemes lies in the fact that in the parallel approach, the islands execute similar algorithms to the ones used in a corresponding sequential implementation.
In island-based models the specification of the migration stage is particularly important because it allows the cooperation among islands. Therefore, the migration policy must be carefully designed to ensure the best possible convergence. In [49] some of the most common migration stages were analysed. In order to configure the migration stage, it is necessary to establish the migration topology (where to migrate the individuals) and the migration rate (how many individuals - $|Inm|$ - are migrated and how often). In synchronous schemes migrations are performed at fixed generations. In asynchronous models a probability of migration is established. Each time a generation finishes, a migration is performed with such a probability. In this thesis, asynchronous models have been used. In addition, individuals which are going to be migrated and those which are going to be replaced must be selected. Such a selection is performed by the use of the migration scheme (or selector) and the replacement scheme (or selector), respectively.
Regarding the migration topologies, the next ones have been used:

- All to all connected topology (ALL): In this topology each island connects and sends its individuals to all the remaining ones.
- Ring topology (RING): In this topology each island connects to exactly two other islands constituting a logical ring. Considering that there are n_p islands, labelled from 0 to $n_p - 1$, each island γ sends its individuals to the island $(\gamma + 1) \bmod n_p$, and receives individuals from the island $(\gamma + n_p - 1) \bmod n_p$.

Several migration and replacement schemes have also been tested. In the case of mono-objective approaches, the two most important migration and replacement

schemes are the random and fitness-based strategies [49]. In the random migration scheme, random individuals from the population are selected. In the fitness-based or elitist strategy, the best ones are selected. In the random replacement strategy the individuals to replace in the destination island are randomly selected. Finally, in the fitness-based replacement strategy, the worst individuals of the population are selected. Such schemes can be combined to constitute four different migration stages. A comparison among them was presented in [49].
In the case of multi-objective approaches, the most used migration and replacement schemes are summarised in [240]. The most frequently used migration schemes are the following:

- Random: $|Inm|$ individuals are selected randomly from the population.
- Elitist random (ELI-RAND): $|Inm|$ non-dominated individuals are randomly selected. If the number of non-dominated individuals is lower than $|Inm|$, every non-dominated individual is selected.
- Elitist niching (ELI-NICH): $|Inm|$ non-dominated uniformly distributed individuals are selected. As in the previous case, if the number of non-dominated individuals is lower than $|Inm|$, every non-dominated individual is selected.

The most frequently used replacement schemes are the following:

- Random: The individuals to be replaced are randomly selected.
- None: The population size is increased during a generation, so replacement is not required.
- Elitist random (ELI-RAND): The individuals to be replaced are randomly selected among the dominated individuals. If there are no enough dominated individuals, some of the immigrants are discarded.
- Elitist ranking (ELI): The individuals to be replaced are selected from the worst ranked fronts of the population.
- Elitist 100% ranking (ELI100): The population and immigrants are first combined. Then, the elitist ranking replacement scheme is applied.
- NSGA-II Crowding (NSGA-II-CROWD): The population and immigrants are first combined. Then, the crowding operator is used to select the individuals that survive.

Other replacement schemes are based on selecting individuals that are very similar to the immigrants. The aim of such replacement selectors is to maintain the diversity of the population. Based on such ideas the Elitist Hamming-based replacement scheme (HAM) was proposed for binary-encoded individuals. First, it checks whether or not the immigrant has a higher fitness than all the individuals of the destination island. If so, the immigrant replaces the individual with the lowest Hamming distance to it, considering the decision space. Otherwise, the immigrant is discarded.

Chapter
3

Recent Developments in Optimisation

This chapter is devoted to present a survey of the state of the art of some of the recent developments in optimisation. It is focused on the schemes that have been considered in this research. Specifically, two main topics have been covered: hyper-heuristics, and multiobjectivisation. In both cases the main principles and motivations that guide their designs are presented. In addition, a summary of the main taxonomies, best-known approaches, and main practical applications are exposed. In the case of hyperheuristics both mono-objective and multi-objective schemes have been considered.

3.1 Hyperheuristics

3.1.1 Principles and motivation

Hyperheuristics are a set of approaches which are motivated - in part - by the goal of automating the design of heuristic methods that solve hard computational optimisation problems. In the context of optimisation, the term hyperheuristic was firstly used by Cowling [67]. The main distinguishing feature of hyperheuristics is that they operate on a search space of heuristics or metaheuristics, rather than directly on the search space of solutions to the problem that is being solved [38]. An underlying research challenge is to develop more generally applicable search methodologies. There are two fundamental ideas behind the notion of hyperheuristics:

- The process of selecting or designing efficient hybrid heuristics is a search problem in itself.

- There is a significant potential to improve search methodologies by the incorporation of learning mechanisms that can adaptively guide the search.

These ideas have inspired different types of hyperheuristics, so several methods that automate the usage and/or design of heuristics have been proposed in the literature. There are two clear categories of methods. On the one hand, hyperheuristics based on heuristic selection try to identify and select which among a set of low-level heuristics or metaheuristics are the most appropriate ones to solve a particular optimisation problem instance. On the other hand, the hyperheuristics based on heuristic generation have the aim of automating the generation of heuristics to solve a particular problem. They usually combine simple components to generate a more complex heuristic. Considering it, Burke *et al.* proposed the following definition of hyperheuristic [38].

Definition 11 *A **hyperheuristic** can be defined as a search method or learning mechanism for selecting or generating heuristics to solve computational search problems.*

In this research, several innovations regarding hyperheuristics based on selection of heuristics have been proposed. However, hyperheuristics based on generation of heuristics have not been considered. Therefore, in this dissertation the term hyperheuristic is always used to refer to hyperheuristics based on selection of heuristics. Considering it, a hyperheuristic can be viewed as a heuristic that iteratively chooses between a set of given low-level heuristics or metaheuristics in order to solve an optimisation problem [40]. Hyperheuristics operate at a higher level of abstraction than traditional heuristics because they have no knowledge about the problem domain. The underlying principle in using a hyperheuristic approach is that different metaheuristics have different strengths and weaknesses, and it makes sense to combine them in an intelligent manner. The motivation behind the approach is that, ideally, once a hyperheuristic algorithm has been developed, several problem domains and instances could be tackled by only replacing the low-level heuristics or metaheuristics. Thus, the aim of using a hyperheuristic is to raise the level of generality at which most current heuristic systems operate. In addition, by combining the advantages of each of the involved low-level heuristics, a method that is better than any of the single low-level approaches might be obtained.
A diagram of a general hyperheuristic framework [40] is shown in Figure 3.1. It shows a problem domain barrier between the low level heuristics and the hyperheuristic itself. The data flow received by the hyperheuristic could include the quality of achieved solutions (average, improvement, best, worst), the resources (time, processors, memory) invested to achieve such solutions, etc. Based on such information,

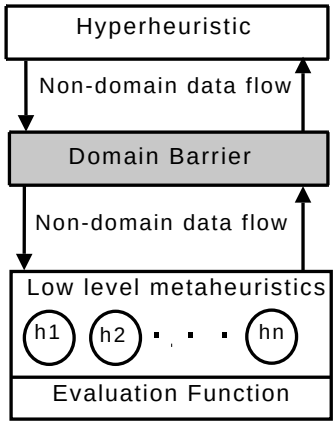


Figure 3.1: Hyperheuristic Framework

the hyperheuristic make its decisions. The data flow coming from the hyperheuristic could include information about which heuristic must be executed, its parameters, stopping criteria, etc.

Hyperheuristics are highly related to the problem of parameter control [219]. Parameter control is a special case of parameter setting that allows changing the parameter values of a metaheuristics during its execution. Several studies [91] have concluded that the usage of static set of parameters during a metaheuristic run seems to be inappropriate. The main drawback is that there is no guarantee that a fixed set of parameters leads to optimal performance. Since different parameter settings are needed to emphasise either exploration or exploitation, it follows that optimal parameter settings might vary over time. In fact, it has been empirically and theoretically demonstrated that different values of parameters might be optimal at different stages of the optimisation [14, 223]. Thus, the aim of parameter control is to design a control strategy that selects the parameters to use at each stage of the optimisation.

Several strategies that adapt the parameters of the algorithms have been designed. The control strategies can be deterministic, adaptive or self-adaptive [179]. Deterministic strategies modify the parameter values by using a deterministic rule, i.e. with a fixed schedule. Adaptive parameter control aims to modify the parameter values based on some feedback from the search behaviour. Finally, self-adaptive parameter control encodes the parameter values into the solutions and they are subject to variations following the rules of the metaheuristic in which they are used. Hyperheuristics are popular general methods which can be applied to perform an adaptive parameter control. The main advantage of hyperheuristics is that they are independent of the metaheuristics that they control.

Considering the case of parameter control, hyperheuristics provide two main benefits. First, in the cases where a specific parameterisation is superior to the others in every stage of the optimisation, the hyperheuristic might automatically detect it. Thus, the user effort associated to the parameter setting might be decreased. In addition, different values of parameters might be optimal at different stages of the optimisation process [134]. In such cases, hyperheuristics might grant more resources to the fittest low-level configurations on each stage of the optimisation process. Therefore, the quality of the solutions obtained by the hyperheuristic might be higher than the quality of the solutions obtained by any of the involved low-level configurations. On the other hand, the goal of raising the level of generality is achieved at the expense of reduced - but still acceptable - solution quality when compared to tailor-made approaches.
Finally, it is important to remark that the aim of hyperheuristics goes beyond the objectives of parameter control because hyperheuristics might also make possible the integration of several different metaheuristics with different characteristics. Hyperheuristics have been successfully applied in several problem domains [38]. This proves the generality of the designed schemes. Some of the tested domains have been: production scheduling, cutting and packing, and vehicle routing.

3.1.2 Classification

Several classifications of hyperheuristics approaches have been proposed in the literate. In [222] two main categories were identified: hyperheuristics without learning and hyperheuristics with learning. Hyper-heuristics without learning use several heuristics according to a predetermined sequence. They are mainly based on the fact that when a heuristic is blocked on a local optimum, the usage of another heuristic might avoid the stagnation. Therefore, changing the used heuristics in a systematic way might lead to a better performance. On the other hand, hyperheuristic with learning selects the low-level approaches based on some feedback from the search behaviour. In such cases, the hyperheuristic selects a promising low level approach at each decision point based on the information about the effectiveness of each low level heuristic accumulated in earlier stages of the execution (or in previous runs). The aim is to assign more resources to the most promising heuristics, so that better results can be obtained. In [20, 36] the categories in which the hyperheuristics are classified are based on the features of the used low-level heuristics. Specifically, a distinction between those which are constructive and those which are local search methods is proposed. In [17] the aim of the hyperheuristic is used to establish the categorisation. It distinguishes between hyperheuristics whose aim is to choose heuristics, and hyperheuristics whose aim is to build heuristics. Finally,

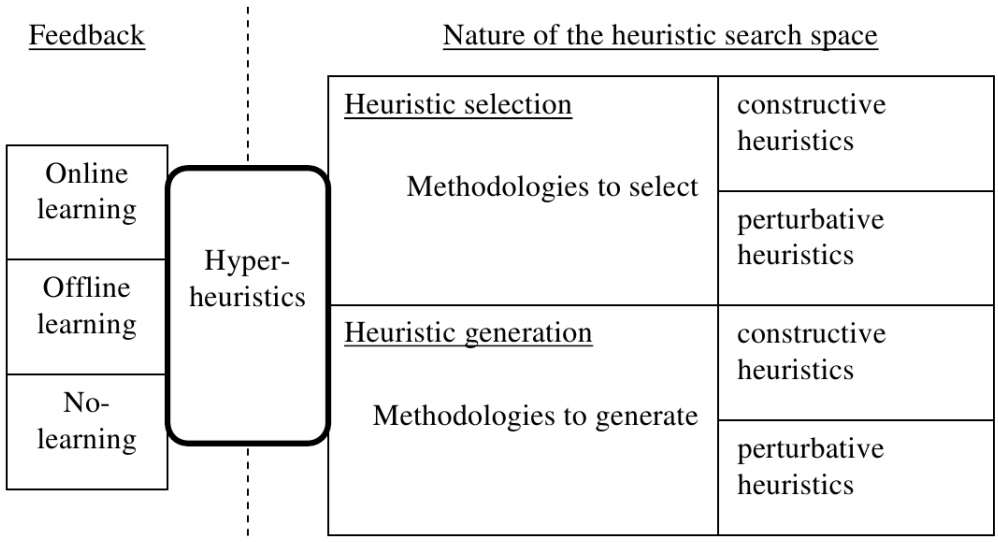


Figure 3.2: Classification of hyperheuristics proposed by Burke *et al.*

in [51] four categories are identified:

- Hyperheuristics based on the random choice of low level heuristics.
- Greedy and peckish hyperheuristics.
- Metaheuristic-based hyperheuristics.
- Hyperheuristics employing learning mechanisms to manage low level heuristics.

The classification proposed by Burke *et al.* [39] is a unifying view of the ones previously presented. The authors use two dimensions to classify the hyperheuristics: the nature of the heuristic search space, and the source of feedback. Figure 3.2 shows a diagram of the considered classification. The dimensions are considered to be orthogonal, i.e., different heuristic search spaces can be combined with different sources of feedback.
According to the nature of the heuristic search space, the hyperheuristics can be classified into two clear categories: heuristic selection and heuristic generation. Moreover, a second level in this dimension corresponds to the distinction between constructive and perturbation hyperheuristics. This categorisation is concerned with the nature of the low-level heuristics. Thus, constructive hyperheuristics are the ones that employ constructive low-level heuristics. A constructive heuristic build a

solution incrementally. It starts from an empty solution, and at each step a new component is added to the solution. On the other hand, perturbation hyperheuristics (also named improvement hyperheuristics) are the ones that employ perturbation or improvement heuristics. A perturbation heuristic starts with a complete solution, or with a set of complete solutions. At each step, the heuristic try to improve the set of current solutions by perturbing them.
Finally, according to the source of feedback, three different categories have been identified:

- *Online learning hyperheuristics* are the ones that learn while solving a given instance of a problem.
- *Offline learning hyperheuristics* are the ones that learn from a set of training instances prior to be applied to a new unsolved instance.
- *No-learning hyperheuristics* are the ones that do not use feedback from the search process. They usually apply a fixed schedule. Since cooperative search methods integrate several heuristics, they might be considered in this group.

3.1.3 Mono-objective hyperheuristics

Most of the research in hyperheuristics has been conducted with mono-objective optimisation problems. Previously to the appearance of the concept of hyperheuristic, several researchers analysed similar concepts, but using different nomenclatures. In fact, the idea of automating the selection of good heuristics and the adaptive setting of parameters can be traced back to the 1960s and 1970s. The principles of hyperheuristics have emerged from researches in several fields that considered mono-objective approaches [38]:

- Automated heuristics sequencing [100, 150].
- Automated planning systems [116].
- Automated parameter control in EAs [71].
- Automated learning of heuristic methods [181].
- Automated prioritising [136].

Since the first usage of the term hyperheuristic [67], several different implementations have been proposed. Some of them make use of constructive low-level approaches, while some of them operate with perturbative techniques. In addition, most of the

techniques operate as single-point schemes, i.e. at each stage of the optimisation a single solution is managed. However, some multi-point search schemes have also been proposed.

Considering the single-point schemes that operate with perturbative heuristics, one of the most-known proposals is the tabu search based hyperheuristic proposed by Burke *et al.* [41]. In such a case, each low-level heuristic is a definition of a neighbourhood. They are scored based on the previous improvements with a reinforcement learning scheme. In addition, when a neighbourhood do not produce improvements, it is inserted into a tabu list. The same hyperheuristic was used with SA [84]. Offline learning methods has also been proposed [233]. The low-level heuristic used at each stage of the optimisation depends on the solutions obtained with other instances. Alternatively, several methods [67, 139] select the low-level approaches in very simple ways and decide to accept or reject the changes performed by the heuristics depending on the obtained fitness values. In some cases the acceptance strategy is probabilistic [13]. Finally, the choice functions, are a very popular technique [67, 138]. In such cases, a scoring function is used to assess the performance of each low-level heuristic. The resources are granted to the low-level heuristic that maximises such a function. The choice function is the sum of several components. Some of the components intensify recent performance, while other components provide diversification. In addition, the choice function approaches also make use of memory. The methods that are unsuccessfully applied with a given solution are not used again until the scheme escapes from such a solution. One of the main drawbacks of the approach is that it requires the setting of several parameters that highly depend on the problem which is being solved. A more robust approach based on choice functions was used in [64, 65, 66]. In such proposals the parameters of the choice functions are dynamically adapted, so no parameters are required. Finally, in [241], a choice function was also used to score each method. However, the resources were assigned using a probability function which is based on the assigned score. In every case, the choice functions have been used with single-point search methods, and they have been used to select between different neighbourhoods.

Some single-point schemes that operate with constructive techniques have also been proposed. One of the most known schemes is the proposal of Ahmadi *et al.* [5]. It uses a VNS, but the search is in the heuristic space. Other metaheuristics have also inspired the creation of this kind of hyperheuristics. A tabu search based hyperheuristic was proposed in [43]. Finally, hill-climbing based hyperheuristic have also been proposed [11]. It applies a restart mechanism to avoid local optima.

The multi-point strategies that operate with perturbative schemes are not so extensive. Cowling proposed the usage of an indirect GA [63]. Each individual in the population gives a sequence of heuristic choices that indicates which low level

heuristic to apply at each step. The proposal was extended to incorporate adaptive length chromosomes [120]. Similar approaches but considering the ACO algorithm were proposed in [37, 56]. Considering the cooperative search methods as parallel hyperheuristics without learning, several schemes have been proposed [68]. However, in such schemes there is no learning. Therefore, the research in such a topic is no so extensive.
Finally, some multi-point schemes that operate with constructive heuristics have also been proposed. Most of them are based on using EAs [210]. As with the perturbative schemes, each individual in the population gives a sequence of heuristic choices that indicates which low level heuristic to apply at each step. However, in these cases the methods are constructive.
In addition, some hybrid schemes have also been proposed. In [107], a single-point hill-climbing scheme is applied. Some of the low-level heuristics are perturbative, while some of them are constructive techniques. Moreover, some parallel approaches have been proposed in work-in-progress papers [26, 206]. However, they have not been extensively tested.

3.1.4 Multi-objective hyperheuristics

The proposals of hyperheuristics for multi-objective optimisation are not so extensive [38]. Considering the schemes that operate with constructive techniques, the proposals are straightforward extensions of mono-objective techniques. Most of them are based on considering a well-known MOEA with the same individual representation as in mono-objective hyperheuristics [72, 237]. They usually operate with the NSGA-II approach. However, in some cases, other multi-objective algorithms have also been tested [195].
In the case of multi-objective hyperheuristics that operate with perturbative low-level heuristics, the first approach was proposed by Burke *et al.* [42]. The approach was further developed in [44]. It is an extension of the tabu search scheme proposed for mono-objective optimisation [41]. The proposal identifies which are the most appropriate neighbourhoods for each of the considered objectives, i.e. each one of the objective are independently analysed. Then, at certain points during the search, the approach selects the most appropriate neighbourhood heuristic in order to push the solution in the desired direction. Therefore the idea is to choose at each iteration the heuristic that is suitable for the optimisation of a given individual objective. Several different methods for selecting the desired direction were proposed.
Finally, it is important to remark that, since our knowledge, at the beginning of this research there were no any multi-point hyperheuristic for multi-objective optimisation problems.

3.2 Multiobjectivisation

3.2.1 Principles and Motivation

Multiobjectivisation is the process of reformulating a mono-objective optimisation problem as a multi-objective one. Then, it is solved with a multi-objective method in order to provide a solution to the original mono-objective problem. The idea of transforming a mono-objective problem into a multi-objective one was firstly proposed by Louis and Rawlins in 1993 [160]. They demonstrated the advantages of using Pareto-based selection for a deceptive mono-objective problem. Since then, there was no much attention to such an idea for almost a decade. Knowles *et al.* continued the research in 2001 [146]. The authors demonstrated that the decomposition of the original objective into several components might remove some local optima. It was the first time that the term multiobjectivisation was used. A small segment of researchers have explored the inner workings of this new technique in the years since its introduction in 2001. Thus, several proofs, principles and methods have emerged [158].

Research has uncovered several ways in which multiobjectivisation leads to improved performance [158]. First, multiobjectivisation can escape from local optima through avoiding confounding interactions within the search space. The process is called the *breakage of epistasis*. It implies that multiobjectivisation allows traversing regions of the space that would have been otherwise difficult to traverse. In addition, multiobjectivisation might increase the recognition of important fitness improvements. Changes in the variable space might produce an improvement in one of the new generated objectives. Such improvement might be easily detected with multi-objective techniques, but not with mono-objective approaches. Finally, multiobjectivisation might consider the preservation of diversity as an objective. It is known that diversity is a key issue in the performance of EAs, and other population-based approaches [235]. Since multiobjectivisation affects the way in which diversity is maintained, it might lead to improve the obtained solutions.

It has been demonstrated that the usage of multiobjectivisation makes easier the resolution of optimisation problems in some domains. For instance, in [192] multiobjectivisation was successfully used to solve single-source shortest paths problems. However, it can also produce harder problems [34, 121]. Regarding the optimisation approaches used for multiobjectivised problems, initially some simple hill climbers were tested. For instance, in [146] a Pareto Hill Climber was used, while in [121] the performance of several multi-objective hill climbers were analysed. However, more complex methods as MOEAs, which can be effective in overcoming local optima, have reported better solutions in some domains [157].

3.2.2 Best-known Approaches

Multiobjectivisation can be carried out following two general schemes: decomposition and aggregation. The first one is based on decomposing the original objective into several components. Then, each one of the components is used as an objective. Thus, the main objective is only implicitly represented through its decomposed parts, which become objectives. The Pareto Front of the new defined problem should contain a solution with the original optimal value. Usually, the sum of the considered components is equal to the original objective. This ensures that the Pareto Front contains the optimal mono-objective solution. This approach was firstly tested in [146]. A more complete analysis was presented in [121]. It has also been used to solve complex practical optimisation problems [137].
The second type of multiobjectivisation is based on adding a new objective function. Such a function is used in conjunction with the original fitness function. The incorporated objectives are usually referred to as helper-objectives [135]. The helper-objectives guide the search towards solutions containing good building blocks, and help the algorithm to escape from local optimal [159]. It is important to remark that, since the original fitness function is kept as an objective, the Pareto Front of the multiobjectivised problem always contains a solution with the original optimal value. Helper-objectives can be defined as novel objectives [117] or as decomposed parts of the main objectives [135]. In addition, they can be generated by considering problem-dependent [245] or problem-independent information [188]. The main advantage of the second approach is its generality. The considered helper-objectives usually depend solely on the own individual. However, in some novel schemes the added objectives also depend on other individuals of the population or archive [55, 188].
In the cases where problem-independent objectives are considered, the aim is usually to better preserve the diversity. In such cases, multiobjectivisation usually decreases the selection pressure of the original approach. Therefore, when used in combination with EAs or other population-based metaheuristics, some low quality individuals could survive in the population with a higher probability. However, if properly configured, in the long term these individuals could help to avoid stagnation in local optima, so higher quality solutions might be obtained.
Several options have been proposed to define problem-independent helper-objectives [3, 35, 234]. Some of them are based on using the Euclidean distance on the decision space as a direct measure of the diversity:

- Distance to Closest Neighbour (DCN): The distance to the closest neighbour of the population has to be maximised.

- Average Distance to Individuals (ADI): The average distance to all individuals of the population has to be maximised.
- Distance to Best Individual (DBI): The distance to the best individual of the population, i.e. the one with the highest fitness, has to be maximised.

Other authors propose the usage of objectives which may preserve diversity without using a direct measure for diversity [3]. Among them, the following ones have been defined:

- Random: A random value is assigned as the second objective to be minimised. Smaller random values may be assigned to some low-quality individuals that would get a chance to survive.
- Inversion: In this case, the optimisation direction of the original objective function is inverted and is used as the helper-objective. This approach highly decreases the selection pressure.
- Time stamp: The helper-objective is calculated as a time stamp of when an individual is generated. Each individual in the initial population is stamped with a different time stamp represented by a counter which gets incremented every time a new individual is created. From the second population all new generated individuals get the same time stamp that is set to the population size plus the generation index. This time stamp must be minimised.

Note that among these options, DCN and ADI take more time to calculate than the others, because they need to look at $O(N^2)$ distances. Since in real-world applications the evaluation time is usually high, the time required to calculate distances is usually negligible compared to the time required to evaluate the candidate solutions. In [35], the behaviour of several problem-independent helper-objectives in dynamic environments was analysed. It revealed the superiority of the distance-based helper-objectives.

Part II

Problem-Independent Proposals and Validation

Chapter
4

General Algorithmic Proposals

This chapter is devoted to present the main problem-independent algorithmic proposals of this research. The developments can be divided into two groups. The first group consists of a set of novel sequential and parallel hyperheuristics. Proposals for mono-objective and multi-objective optimisation problems have been developed. The second group consists of a set of innovations for multiobjectivisation. The proposals of the second group make use of the new designed hyperheuristics.

4.1 Innovation in hyperheuristics

Several hyperheuristics have been proposed in the literature. However, few researches that consider multi-point perturbative low-level approaches have been conducted. In fact, only some papers have been found for mono-objective optimisation. Most of them have been used to select among different local search neighbourhoods. Moreover, to our knowledge, no research has been conducted for multi-objective optimisation problems. According to the taxonomy proposed by Burke *et al.* [38], the hyperheuristics proposed in this research are online, based on selection of heuristics, and they work with perturbation strategies. Different approaches have been defined for dealing with mono-objective and multi-objective optimisation problems. The same ideas have been behind the design of both kinds of approaches.

4.1.1 Mono-objective hyperheuristic

The proposed mono-objective hyperheuristics are based on the usage of choice functions [67]. In such hyperheuristics, a scoring function is used to assess the performance of each low-level configuration. The low-level configurations are single-point

perturbative schemes. At each decision point the resources are granted to the low-level configuration that maximises such a function. In addition, the choice function approaches also make use of memory. The methods that are unsuccessfully applied with a given solution are not used again until the scheme escapes from such a solution. In order to avoid the usage of additional parameters, the internal parameters might be dynamically adapted [64]. The adaptation is based on modifying the capabilities of intensification or diversification depending on whether the low-level approaches are able to improve on the current solution or not. Choice functions have been traditionally used for selecting among a set of neighbourhood definitions. The adaptation was based in the fact that in some cases local optima might be found. In such stages, the parameters of diversification were reinforced, while in other stages, intensification was reinforced.

In this research, the desire has been to select among multi-point perturbative metaheuristics. Experimentally, it was discovered that in many problems independently of the quality of the metaheuristics, they usually improved on the initial solutions. Therefore, using a tabu memory to avoid the usage of methods unsuccessfully applied in previous stages made no sense. In addition, due to this behaviour, the way in which the parameters were controlled in the original choice functions was not adequate. Therefore, low-quality solutions were found with the original choice functions methods. For this reason, new hyperheuristics based on the idea of choice functions, but considering the aforementioned drawbacks, were defined. Some of the new schemes have incorporated the idea of using probabilistic selection schemes [241].

The first defined hyperheuristic (MONO_WEIGHT) is based on using a scoring strategy and a probabilistic selection strategy for picking the low-level configuration that must be executed. Once a strategy is picked, it is executed until a local stopping criterion is achieved. Then, another low-level configuration is selected, and it is executed taking as initial population the last population of the previously selected approach. This process continues until a global stopping criterion is reached.

At the beginning of the execution, each low-level strategy is executed one time. The order in which they are selected is randomly determined. After this initial stage, the selection of the low-level strategy that must be executed operates as follows. First, the scoring strategy assigns a score to each low-level configuration. This score estimates the improvement that each low-level metaheuristic or configuration can achieve when it starts from the currently obtained solutions. In order to perform such an estimate, the previous fitness improvements achieved by each configuration are used. The improvement (imp) is defined as the difference, in terms of the fitness value, between the best achieved individual and the best initial individual. Considering a configuration $conf$, which has been executed j times, the score - $s(conf)$ - is calculated as a weighted average of its latest k improvements (Equation 4.1). In

such an Equation, $imp[a][b]$ represents the improvement achieved by the configuration a in its execution number b. Depending on the value of k, the adaptation level of the hyperheuristic can be set. The weighted average assigns greater importance to the latest executions.

$$s(conf) = \frac{\sum_{i=1}^{min(k,j)} (min(k,j) + 1 - i) \cdot imp[conf][j-i]}{\sum_{i=1}^{min(k,j)} i} \tag{4.1}$$

The stochastic behaviour of the involved low-level metaheuristics may lead to variations in the results achieved by them. Therefore, it might be appropriate to make some selections based on a random scheme. The hyperheuristic can be tuned by means of the parameter β, which represents the minimum selection probability that should be assigned to a low-level configuration. Being n_h the number of involved low-level configurations, a random selection following a uniform distribution is performed in $\beta \cdot n_h$ per cent of the cases. In the rest of the cases, a selection probability proportional to the obtained score is assigned to each low-level configuration. Therefore, the probability of selecting each configuration $conf$ is given by:

$$prob(conf) = \beta + (1 - \beta \cdot n_h) \cdot \left\lfloor \frac{s(conf)}{\sum_{i=1}^{n_h} s(i)} \right\rfloor \tag{4.2}$$

An additional way of selecting the low-level approaches has been considered. In such a case, the selection probability is not proportional to the obtained score. Instead, the low-level approach that maximises the scoring function is picked. As in the previous case, a random selection following a uniform distribution is performed in $\beta \cdot n_h$ per cent of the cases. This hyperheuristic is named ELI-MONO_WEIGHT.
Prior to the experimental analysis, a theoretical analysis of the approach was performed. In order to facilitate the analysis it was initially performed considering low-level approaches with deterministic behaviours. However, since the hyperheuristics were going to be applied with stochastic configurations, the implications of using stochastic configurations were also analysed. In addition, with the aim of facilitating the analysis, it was assumed that no random assignments were performed ($\beta = 0$). Although very simple approaches were considered, important conclusions could be drawn. Such study is presented in the following lines.

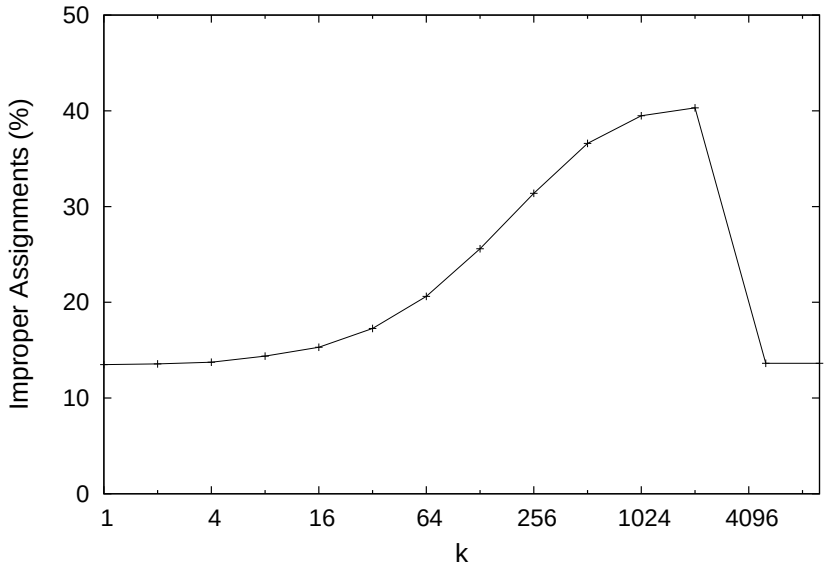


Figure 4.1: Assignments to ConfBad for the trivial problem

First, a trivial optimisation problem and two artificial low-level configurations were defined. The problem has only one variable (x_1), and the function to minimise is calculated as $f(x_1) = x_1$. The first low-level configuration ($ConfGood$) makes the transformation $newx_1 = x_1 - \frac{x_1}{100}$. The second one ($ConfBad$) makes the transformation $newx_1 = x_1 - \frac{x_1}{1000}$. Hence, $ConfGood$ behaves better than $ConfBad$, for any input. Note that, the best assignment of resources lies in granting all the resources to $ConfGood$.

In order to analyse the impact produced by the parameter k, the hyperheuristic MONO_WEIGHT was executed with such a problem and configurations with several values of k. Figure 4.1 shows the percentage of resources granted to $confBad$ for each value of k. It can be appreciated that the intermediate values of k are the worst-behaved values. Figure 4.2 shows, for the worst value of k (2048), the evolution of the assignments to both configurations. At the beginning most of the resources are granted to $confGood$. However, in the long term, the differences among the granted resources are not so big. The reason is that, when the historical knowledge begins to be discarded, the data saved for the different configurations corresponds to very different stages of the optimisation. In such stages, the hyperheuristic get confounded and many resources are assigned to $confBad$ - note the changes in the slopes after 2500 assignments. In the cases where small values of k are used, every data belongs to the same stage - or at least they are discarded quickly - so this

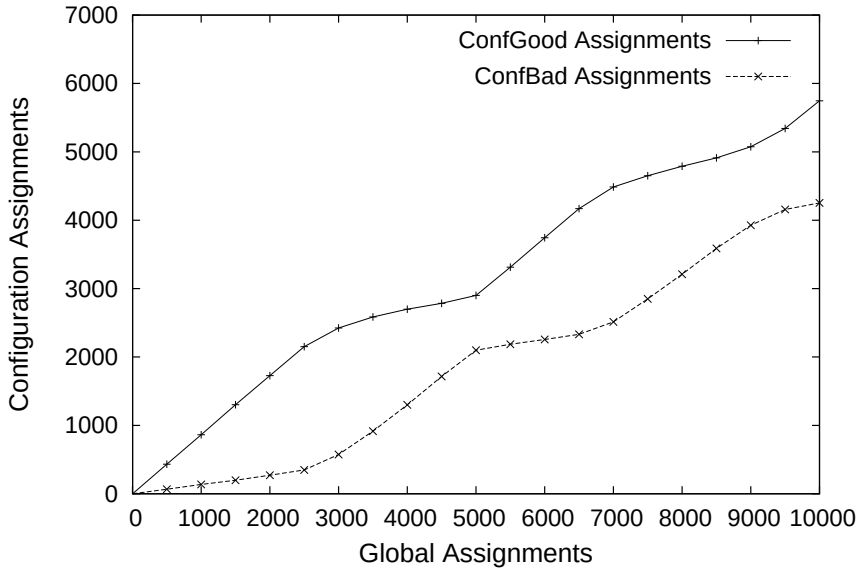


Figure 4.2: Evolution of assignments for the trivial problem (k = 2048)

problem does not appear. Finally, when k is set to a very large value, no data is discarded, so the effect is also minimised. Therefore, with the given hyperheuristics, it seems that it is more likely to obtain good quality solutions by using very large or very small values of k.

In some problems it might happen that the relative performance of the approaches depends of the stage of the optimisation. For instance, considering two low-level approaches ($conf_1$ and $conf_2$), it might happen that initially $conf_1$ is superior to $conf_2$, while when a given fitness is achieved, $conf_2$ begins to behave better than $conf_1$. In such cases, using low-values of k is preferable because when using large values of k the hyperheuristic would assign the resources to $conf_2$ for a larger period. Therefore, using $k = 1$ seems appropriate in such cases. In fact, note that in such cases, the hyperheuristic might get better values than any of the considered low-level approaches.

However, when stochastic approaches are considered, using low values of k might produce some drawbacks. For instance, consider again a case with two low-level configurations. The first configuration $conf_1$ produces a very high improvement with a very high probability, but in some corner cases it does not produce any improvement. The second configuration always produces a very low improvement. In such a case, if $k = 1$ is considered, the resources might not be granted correctly. The reason is that after the appearance of a corner case, all the resources will be

granted to $conf_2$. This would not happen for larger values of k. If the probability of appearance of corner cases is not very low, the value of k should be increased. Therefore, depending on the features of the problem and on the features of the low-level configurations, the best-behaved value of k is expected to change.
On the other way some analyses regarding ELI-MONO_WEIGHT were also performed. In general, the approach MONO_WEIGHT tends to distribute the resources among more configurations, while ELI-MONO_WEIGHT tends to focus on few configurations. For instance, consider a case with two configurations. The first configuration ($conf_1$) always produces an improvement equal to 90, while the second one ($conf_2$) always produce an improvement equal to 10. In such case, the ELI-MONO_WEIGHT would assign all the resources to $conf_1$, while MONO_WEIGHT would assign the 90% of the resources to $conf_1$. Thus, in such a case, the elitist selection scheme is more appropriate. However, in other cases, using a probabilistic selection might be more adequate. For instance consider a case in which the first configuration usually behaves correctly, but in some cases it attains a poor improvement. In addition, the second configuration usually behaves poorly, but in some cases it behaves better than the first configuration. In such a case, ELI-MONO_WEIGHT might get confounded and might assign most of the resources to the second configuration. On the other hand, the MONO_WEIGHT approach would distribute the resources between both configurations, probably assigning more resources to the first configuration. Thus, in cases where the hyperheuristic might get confounded, since ELI-MONO_WEIGHT tends to focus on fewer configurations, it might end with very poor quality solutions.
This analysis shows that depending on the features of the problems, and on the features of the low-level approaches, the adequate selection strategy and the adequate adaptation level - value of k - varies. In fact, it can be found corner cases in which even a random selection scheme might be better than the assignment performed by the hyperheuristic. This could happen for instance if the behaviour of each low-level configuration in a given stage is not useful to determine its behaviour in posterior stages. However, this is not likely to happen with the metaheuristics because usually the changes in the behaviour of the approaches are not so abrupt. Therefore, an experimental evaluation of the designed approaches is mandatory to analyse its behaviour with typical optimisation problems and typical metaheuristics.

4.1.2 Multi-objective hyperheuristic

In the case of multi-objective optimisation, measuring the quality of the solutions is more complex. As it has been described, there are several metrics that have been devised for assessing the quality of the obtained solution sets. The hypervolume metric is one of the most extended approaches. One of its main strength is that it is

a Pareto-compliant metric. Moreover, the knowledge of the Pareto Front of the problem is not required to perform the calculation. For this reason, the MONO_WEIGHT hyperheuristic was extended to muli-objective optimisation, substituting the improvements in fitness by improvements in hypervolume. In order to calculate the improvements the following procedure is used. First, when a low-level approach begins to execute, the hypervolume of its initial population is calculated. In order to calculate the hypervolume a reference point is required. In this research a reference point with all its coordinates equal to 1 have been used. Prior to the calculation of the hypervolume, all the solutions are normalised considering a set of coordinates given by the practitioner. If some solutions are worse than the given normalisation point, they are discarded. The previous operation assumes minimisation. In the cases of objectives that must be maximised, they are inverted. The same procedure is used to calculate the hypervolume at the end of the execution. The difference between the calculated values is stored as the obtained improvement.

Therefore, in the proposed multi-objective hyperheuristic (HV_WEIGHT) the scoring strategy estimates the improvement in hypervolume that each low-level metaheuristic or configuration can achieve when it starts from the currently obtained solutions. In order to perform such an estimate, the previous hypervolume improvements achieved by each configuration are used. Considering a configuration $conf$, which has been executed j times, the score - $s(conf)$ - is calculated as a weighted average of its latest k hypervolume improvements (Equation 4.3). In such an Equation, $hvImp[a][b]$ represents the hypervolume improvement achieved by the configuration a in its execution number b. Depending on the value of k, the adaptation level of the hyperheuristic can be set. As in the mono-objective case, the weighted average assigns greater importance to the latest executions.

$$s(conf) = \frac{\sum_{i=1}^{min(k,j)} (min(k,j) + 1 - i) \cdot hvImp[conf][j-i]}{\sum_{i=1}^{min(k,j)} i} \tag{4.3}$$

The stochastic behaviour of the involved low-level metaheuristics may lead to variations in the results achieved by them. Therefore, it might be appropriate to make some selections based on a random scheme. The same approach as in the mono-objective case has been used. Therefore, the probability of selecting each configuration $conf$ is given by:

$$prob(conf) = \beta + (1 - \beta \cdot n_h) \cdot \left[\frac{s(conf)}{\sum_{i=1}^{n_h} s(i)} \right] \tag{4.4}$$

An additional way of selecting the low-level approaches has been considered. In such a case, the resources are not granted proportionally to the obtained score. Instead, the low-level approach that maximises the scoring function is picked. As in the previous case a random selection following a uniform distribution is performed in $\beta \cdot n_h$ per cent of the cases. This hyperheuristic is named ELI-HV_WEIGHT.

4.1.3 Dynamic-mapped Island-based Model

The previous hyperheuristics has been parallelised by integrating the approach in an island-based model. The basic operation of the island model remains intact. Therefore, in the designed scheme, the overall population is divided into a number of independent and separate subpopulations which are associated to an island. Each island evolves in isolation for the majority of the execution - using a population-based metaheuristic-, but occasionally, some individuals are migrated between neighbour islands.
In the standard island-based model, there exists a static mapping among the islands and configurations, i.e., each island executes the same configuration during the complete run. In a homogeneous island-based model, there is only one configuration that is executed by every worker island. In a heterogeneous island-based model, the configurations executed by worker islands are different. However, in the new designed model, a dynamic mapping among the islands and configurations is applied. Thus, configurations executed in each island vary during the run. This mapping is performed using the new designed hyperheuristics. When a new configuration is selected for executing in an island, it continues with the last found population. This might produce some drawbacks in the cases when approaches without elitism are used. For such cases, an archive is kept in the island. Therefore, the new selected configuration can start with high-quality solutions.
In order to manage the dynamic mapping, i.e., to apply the hyperheuristic principles, a new special island, called *master island*, is introduced into the scheme. In order to implement it, two kinds of stopping criteria are defined. First, a global stopping criterion is established. This is similar to the one established in the standard island-based model. When this global stopping criterion is reached, every worker island sends its local solution to the master and the run ends. Moreover, a local stopping

criterion is set for the execution of the configurations on the worker islands. When a local stopping criterion is reached, the island execution is stopped. Then, the local results are sent to the master island. At this point, the master island applies the hyperheuristic with the aim of deciding which low-level configuration is going to be executed in the idle island.
An additional change was performed with the aim of adapting the hyperheuristic to parallel environments. In island-based models, the migration stage might completely change the quality of the considered populations. Since the hyperheuristic evaluates the performance of the approaches by considering the members of the population, it must take into consideration the effect caused by the immigrants. Specifically, the improvements obtained during the migration stage are measured and they are deducted from the overall obtained improvement. In this way, the migration stage is kept, but the effect over the hyperheuristic is minimised.
It is also important to note that in this work, the same stopping criterion has been established in every island. This facilitates the operation of the hyperheuristic, because it does not need to consider the time used by each low-level approach. In addition the hyperheuristic have completely ignored the features of the processors used in the islands. In the cases where a homogeneous environment is used, this is not a problem. However, in heterogeneous cases the resources might not be assigned in a correct way.

4.1.4 Other tested hyperheuristics

Several alternative ways of performing the assignment of resources were tested while developing the previously presented methods. Such approaches did not obtain the quality of the previously detailed hyperheuristics, or suffered from other kind of drawbacks. Although extensive analyses have not been performed with them, they are described here, and their main weaknesses are detailed.
In the case of the mono-objective approaches, two alternative methodologies were tested. The first one is an extension of some of the ideas used with the choice functions of Cowling [67]. In such choice functions there is a component that measures the synergies of using two low-level approaches consecutively. This idea produced a good behaviour in the cases where local search neighbourhood were controlled, probably because the transformations produced by some neighbourhood could be complemented with the ones performed by other ones. However, in the case of controlling metaheuristics the benefits could not be reproduced. In addition, a similar idea was tested with the parallel scheme. In this case, the hyperheuristic (MONO_SYN) tries to detect how well a low-level approach operates in parallel with another approach. MONO_SYN assigns two different scores to each configuration. The

first score is called the independent score (is) and represents the independent performance of each configuration. It is calculated as s in MONO_WEIGHT. The second score is called the cooperation between pairs (cp) and represents the performance of a low-level approach in the presence of other metaheuristics. The improvements achieved during the execution by a metaheuristic m_1, executed in parallel with a metaheuristic $m2$, are saved in the set $imp[m_1][m_2]$. Given two metaheuristics m_1 and m_2, which have been executed in parallel j times, the score $cp(m_1, m_2)$ is calculated as a weighted average of the last k improvements achieved by m_1, in the presence of m_2.

$$cp(m_1, m_2) = \frac{\sum_{i=1}^{min(k,j)} (min(k,j) + 1 - i) \cdot imp[m_1][m_2][j-i]}{\sum_{i=1}^{min(k,j)} i}$$

Given a metaheuristic m_1 and the set of currently assigned metaheuristics $m\_set = \{h_1, h_2, ..., h_n\}$, the score $c_s(m_1)$ is calculated as the maximum cp of any of its components, i.e., $c_s(m_1) = max\{cp(m_1, h_1), cp(m_1, h_2), ..., cp(m_n, h_n)\}$.
When every island is idle, the hyperheuristic must grant the resources among the available configurations. The first assignment is performed as in MONO_WEIGHT, but substituting s by is, i.e., the selection probability of the configuration $conf$ is given by:

$$prob(conf) = \beta + (1 - \beta * n_h) * \left[\frac{is(conf)}{\sum_{i=0}^{n_h} is(i)} \right]$$

For the remaining assignments, the global cooperation c_s is also considered. c_s is used with a probability γ. Thus, considering that hh is the set of configurations assigned to any of the islands, the selection probability of the configuration $conf$ is given by:

$$prob(conf) = \beta + (1 - \beta * n_h - \gamma) * \left[\frac{is(conf)}{\sum_{i=0}^{n_h} is(i)} \right] + \gamma * \left[\frac{c_s(conf)}{\sum_{i=0}^{n_h} c_s(i)} \right]$$

One of the main drawbacks of the hyperheuristic is that it assumes that every island finishes at the same time. This forces to select the same stopping criterion in every island. If the finalisation of one island is delayed, the remaining processors keep idle until it finishes. Moreover, the results obtained by it did not outperform the ones obtained by MONO_WEIGHT, so the higher complexity of the approach is not well-grounded.
Finally, a hyperheuristic based on performing an exponential regression (MONO_EXP) was devised. In this case, at the finalisation of each low-level approach both the improvement, and the fitness of the best individual of the initial population are stored. The score of each low-level approach is estimated by performing an exponential regression considering the stored data, and the best individual of the current population. However, the stochastic behaviour of the low-level approaches might produce high changes in the estimated score. Therefore, for cases in which the variation of the obtained solutions is large, this hyperheuristic is not adequate.
In the case of multi-objective hyperheuristics the hyperheuristics MONO_SYN and MONO_EXP were adapted by substituting the fitness improvements by hypervolume improvements. They suffered from the same drawbacks than the mono-objective approaches. Additionally, a scheme based on the coverage metric was developed. The score was based on calculating how many new non-dominated solutions had been generated by the approach. It obtained good quality solutions when few low-level configurations were used, in some cases. However, the main drawback is that assessing the quality of an approach considering only the number of solutions inserted and not its quality is not adequate in many cases. Therefore, many non-adequate cases were also found.

4.2 Innovations in multiobjectivisations

In this research some new multiobjectivisation schemes have also been devised. Moreover, they have been integrated with the previously presented hyperheuristics. This section is devoted to present the new designed schemes.

4.2.1 Multiobjectivisation with parameters

Several ways of *multiobjectivisation* have been proposed in the literature. In this research an extension of the schemes proposed by Toffolo *et al.* [234] has been proposed. Such kind of schemes belongs to the group where the calculation of the added helper-objective depends on other individuals of the population or archive [188]. Specifically some innovations for the multiobjectivisations based on Euclidean distance have been devised. Such multiobjectivisations are the following ones:

- DCN: The distance to the closest neighbour of the population has to be maximised.
- ADI: The average distance to all individuals of the population has to be maximised.
- DBI: The distance to the best individual of the population, i.e. the one with the lowest fitness, has to be maximised.

Such approaches have been executed with several problems and with different kind of metaheuristics in the literature. They have produced benefits in many cases. However, in other cases, optimising the multiobjectivised problems has been harder. Experimentally, it was identified that in some cases the cause of the poor performance, was that poor-quality individuals were kept in the population just because they were very far from the rest of the individuals in the search space. This can be useful in some cases to maintain a good diversity, but if the quality of the individuals is too low, maintaining them might not produce any benefit and might delay the convergence speed.
Considering such a drawback, but at the same time taking into account the desire of maintaining a good diversity, a new helper-objective was defined. The new approach can be applied with DCN, ADI and DBI. They are named DCN-THR, ADI-THR and DBI-THR. The following explanation is done considering the DCN case.
The new helper-objective is named DCN-THR because it incorporates the usage of a threshold ratio ($th \in [0, 1]$) which must be specified by the user. The threshold ratio is used to avoid the survival of individuals with a very low quality. Being $bestFit$ the fitness value of the best individual of the population, and $shift$ a value that ensures that $bestFit - shift \geq 0$ in the whole execution, the threshold value (v) is defined for minimisation problems as:

$$v = \frac{(bestFit - shift)}{th} + shift \tag{4.5}$$

In the case of a maximisation problem, the threshold value is defined as:

$$v = (bestFit - shift) \cdot th + shift \tag{4.6}$$

The helper-objective of individuals whose fitness value is worse than v is assigned to 0. As a result, individuals that are not able to achieve the fixed threshold are penalised. In the special case where $th = 0$, individuals are never penalised. Thus, DCN-THR with $th = 0$ has the same behaviour than the DCN function.
Figure 4.3 (left-hand side) shows the behaviour of the DBI and the DCN functions when they are integrated with SPEA2 and NSGA-II. The maximisation of the

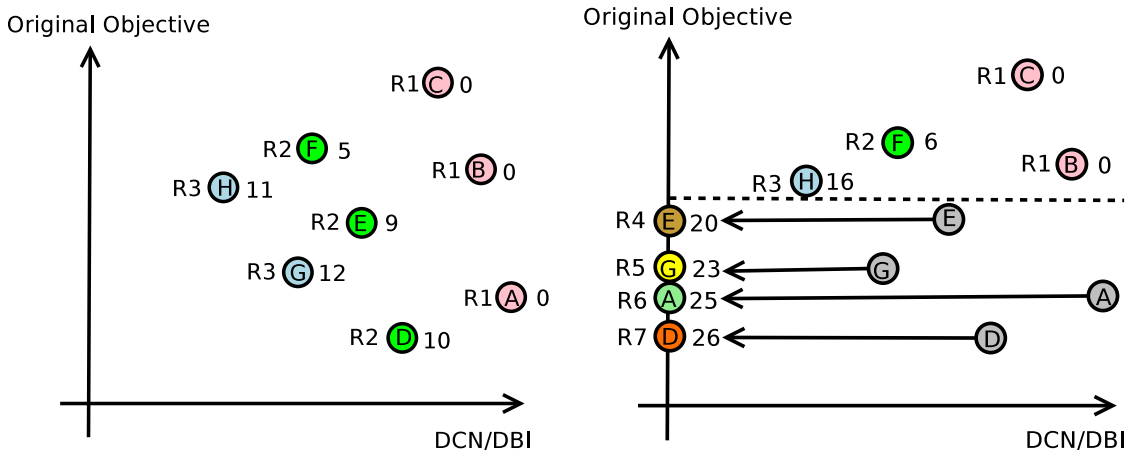


Figure 4.3: Behaviour of the metaheuristics without threshold (left-hand side) and with threshold (right-hand side)

original and the artificial objective functions are assumed. It can be observed that every candidate solution is tagged with a label that indicates its corresponding ranking assigned by NSGA-II (left side of every solution). By this way, the label Ri means that the corresponding candidate solution belongs to the rank number i. Moreover, the corresponding raw fitness assigned by the SPEA2 is also shown at the right side of every candidate solution. The right-hand side of Figure 4.3 shows the effect of incorporating the threshold to the DBI and DCN functions. The broken line represents the value of v. It can be noted that every candidate solution which does not fulfil the minimum quality level established by the threshold ratio is shifted in the objective space. Specifically, a value equal to 0 is assigned to the corresponding alternative objective. The effect of the shift is that the corresponding candidate solution will usually belong to a worse rank in the case of the NSGA-II. In the case of the SPEA2, a highest raw fitness will be assigned to the corresponding candidate solution. Therefore, the survival probability of the candidate solution will be usually decreased. By using an adequate value of th, poor quality individuals can be discarded, but at the same time, the objective of maintaining a high diversity in the population is considered.

4.2.2 Adaptive Multiobjectivisation

The experimental analysis of the new defined helper-objectives shown that in many cases they might produce a high improvement when compared to the original ap-

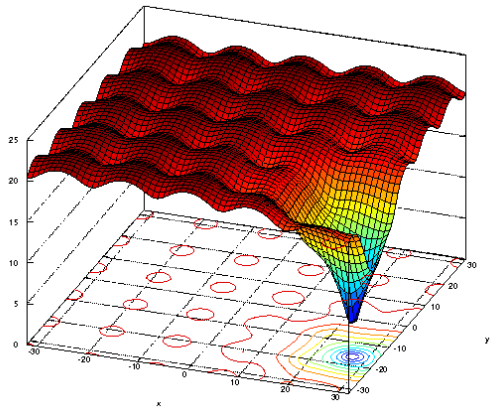


Figure 4.4: Shifted Ackley function with two variables

proach. The behaviour of the threshold is somewhat similar to the one considered in SA. In SA by accepting worsening moves higher quality solutions might be obtained. However, not every worsening move is accepted. Figure 4.4 shows the shifted Ackeley function with two variables. Such a function is multimodal. In the case of using a completely elitist approach that does not accept worsening moves, the execution would probably end with a local optimum. However, with the incorporation of multiobjectivisation, the probabilities of escaping might be larger. The main drawback is that depending on the fitness landscape, the proper values of the threshold vary. For instance, if the deep of the holes had been larger, a low value of th would be preferred. Moreover, the best-behaved values of th might even depend on the stage of the optimisation, because the different parts of the fitness landscape might have different properties. Therefore a fixed value of th might not be adequate.
For this reason, the proposed multiobjectivisation was integrated with the mono-objective hyperheuristics previously presented. In this way, a set of configurations with different values of th might be used as the low-level approaches. In the cases where different values of th are adequate for the different optimisation stages, the hyperheuristic might detect it, and it might vary the way in which the resources are granted during the run. Anyway, even for cases in which the same value is adequate for every stage of the optimisation process, using the hyperheuristic is preferable. The reason is that the user and computational effort might be minimised.

Chapter
5

Validation with Benchmark Optimisation Problems

This chapter is devoted to describe the set of experiments that has been carried out to validate the proper performance of the problem-independent techniques proposed in this research. An analysis of the designed hyperheuristics and multiobjectivisation schemes is presented. The validation is performed with a set of well-known mono-objective and multi-objective benchmark problems.

5.1 Mono-objective Benchmark Problems

5.1.1 Problems Description

Testing mono-objective algorithms in a systematic way is very important. Experimental comparisons between algorithms imply the question of which problems should be used. Three different approaches are usually considered to perform the comparisons among mono-objective solvers [92]:

- Using problem instances from an academic benchmark repository.
- Using problem instances created by a problem instance generator.
- Using real-life problem instances.

The main advantage of using academic benchmarks problems is that the main properties of the optimisation problems are usually known. Therefore, it might be possible to relate the performance of the approach to such features. However, since the number of considered problems is usually not too large, and several features of the problems are ignored, drawing general conclusions is not easy [90]. In addition,

Table 5.1: Main features of F1-F11 problems

Problem	Modality	Separable	Ease dim. by dim.	Range	Optimum
F1	U	Y	Y	$[-100, 100]^D$	-450
F2	U	N	N	$[-100, 100]^D$	-450
F3	M	N	Y	$[-100, 100]^D$	390
F4	M	Y	Y	$[-5, 5]^D$	-330
F5	M	N	N	$[-600, 600]^D$	-180
F6	M	Y	Y	$[-32, 32]^D$	-140
F7	U	Y	Y	$[-10, 10]^D$	0
F8	U	N	N	$[-65.536, 65.536]^D$	0
F9	U	N	Y	$[-100, 100]^D$	0
F10	U	N	N	$[-15, 15]^D$	0
F11	U	N	Y	$[-100, 100]^D$	0

benchmark problems are usually not time consuming - at least compared to real-world problems - so a larger amount of tests can be done. Among the features that are usually considered, the most used ones are the following:

- Modality: An objective function is *multimodal* when it has multiple local optima, while is said to be *unimodal* if it has a single optimum.
- Deception: A function is deceptive if it has at least two optima - a global one, and a local one - and the majority of the search space favours the local optimum.
- Separability: An objective is separable if it can be expressed as product of functions, each one of them depending on one variable.
- Ease of optimisation dimension by dimension, i.e. if the function can be optimised by considering each parameter independently of the others.

There have been several attempts to define test suites or toolkits for building test suites. For instance, the so-called DeJong test suite [73], which consists of five test functions, was very popular during several years. However, some more novel test functions that incorporate additional features have been defined. Among them, some of the most popular ones [90] are the OneMax, the Sphere Model, the Schwefel's function, and the Generalised Rastrigin's function. Recently, in a special issue of

the journal Soft Computing, a test suite with 19 functions was proposed [162]. One of the main advantages is that they are scalable test functions, so the practitioner can select the amount of variables (D) of the optimisation problems. The functions are named F1-F19. The functions F1-F11 are shifted versions of some popular test functions. The functions F12-F19 are combinations of the others. In this work, the experimental comparison with benchmark problems has considered the functions F1-F11. Table 5.1 summarises the main features of the test functions.

5.1.2 Experimental Evaluation of the Mono-objective Hyperheuristics

In this section the experimental evaluation performed with the mono-objective hyperheuristics proposed in this research is presented. Since the adaptive multiobjectivisation presented in this research also uses such hyperheuristics, this analysis is complemented with the one performed in the following sections. The main aim of the experimental evaluation has been to demonstrate that the defined mono-objective hyperheuristics can automatically distribute the computational resources in an intelligent way among a set of low-level heuristics, and to demonstrate that in some cases, the hyperheuristic might even outperform the best low-level considered approaches. In addition, the adaptation level and the different kind of selection schemes have been compared. Finally, several experiments devoted to analyse the behaviour of the parallel version of the hyperheuristic have been performed. In this case, an analysis of the influence that the migration stage has over the performance of the parallel approach has been carried out. The scalability of the parallel proposal has also been analysed. The study has been carried out with the F problems. The number of variables (D) has been fixed to 500.
The same computational environment has been used in every experiment of this chapter. Tests have been run on a Debian GNU/Linux computer with four AMD ® Opteron ™ (model number 6164 HE) at 1.7 GHz and 64 GB RAM. The compiler that has been used is GCC 4.4.5. The optimisation schemes have been implemented using METCO. Finally, it is important to remark that since experiments have involved the use of stochastic algorithms, each execution has been repeated 30 times.

First Experiment: Performance vs. Amount of Low-level Approaches

In order to execute a hyperheuristic, a set of low-level approaches (or configurations) must be defined. The performance of the proposal might depend on the amount of low-level approaches. For this reason, the first experiment has been devoted to relate the performance of the hyperheuristic to the amount of considered low-level

configurations. Experiments with up to 128 low-level configurations have been carried out. The low-level configurations have been based on using the mono-objective EA presented in Chapter 2. The applied survivor selection mechanism has been the GEN-S. A fixed variation scheme has been considered. Specifically, the UM and the SBX operators have been used. A direct encoding based on real-valued genes has been considered. In every case, the probability of crossover has been fixed to 1, while the population size has been fixed to 5. Finally, in order to define several low-level configurations, several values of p_m have been considered.

In order to better inspect the behaviour of the hyperheuristic, the results obtained by the low-level approaches must be analysed. For this reason, the low-level approaches were firstly independently executed. In such executions, the stopping criterion was fixed to a total number of 2.500.000 function evaluations. Figure 5.1 and 5.2 show the median of the fitness obtained with different values of p_m for each of the considered benchmark problems. In every case the low values of p_m have been more adequate. In fact, the obtained functions are monotonically increasing. The reason is that the used mutation is very disruptive, so the usage of low p_m values is more adequate.

In order to facilitate the analysis of the hyperheuristic, the probabilities considered for the low-level configurations have been in the range $[\frac{1}{D}, 1]$. Specifically, in order to define a set of C low-level configurations, C values evenly distributed in the specified range have been considered. In this way, adequate and non-adequate strategies are combined, so it is very easy to check whether the hyperheuristic is assigning the resources correctly or not. It is important to remark that in some cases, the best values of p_m might depend on the stage of the optimisation process. However, since the mutation operator is highly disruptive, this is not expected to happen for the highest values of p_m. In fact, the scheme with $p_m = 1$ is a random search. Anyway, in order to avoid inconsistencies in the analysis, comparisons have not only taken into account the way in which the resources have been granted, but also the obtained quality.

Two adaptive schemes have been considered. The first one assigns the resources using the hyperheuristic MONO_WEIGHT. The local stopping criterion has been fixed to 10.000 evaluations, i.e. each 10.000 evaluations the hyperheuristic selects which low-level configuration must keep executing. In order to decide the values of the parameters of the hyperheuristic some preliminary executions were performed. The value of β was fixed in a way that the 10% of the decisions performed by the hyperheuristic follows a uniform distribution, i.e. $\beta * n_h = 0.1$. The value of k was fixed to 5. The second scheme (RANDOM) has been based on assigning the resources randomly. As in the first scheme, the decisions are taken each 10.000 evaluations. In both cases, the global stopping criterion was fixed to a total number

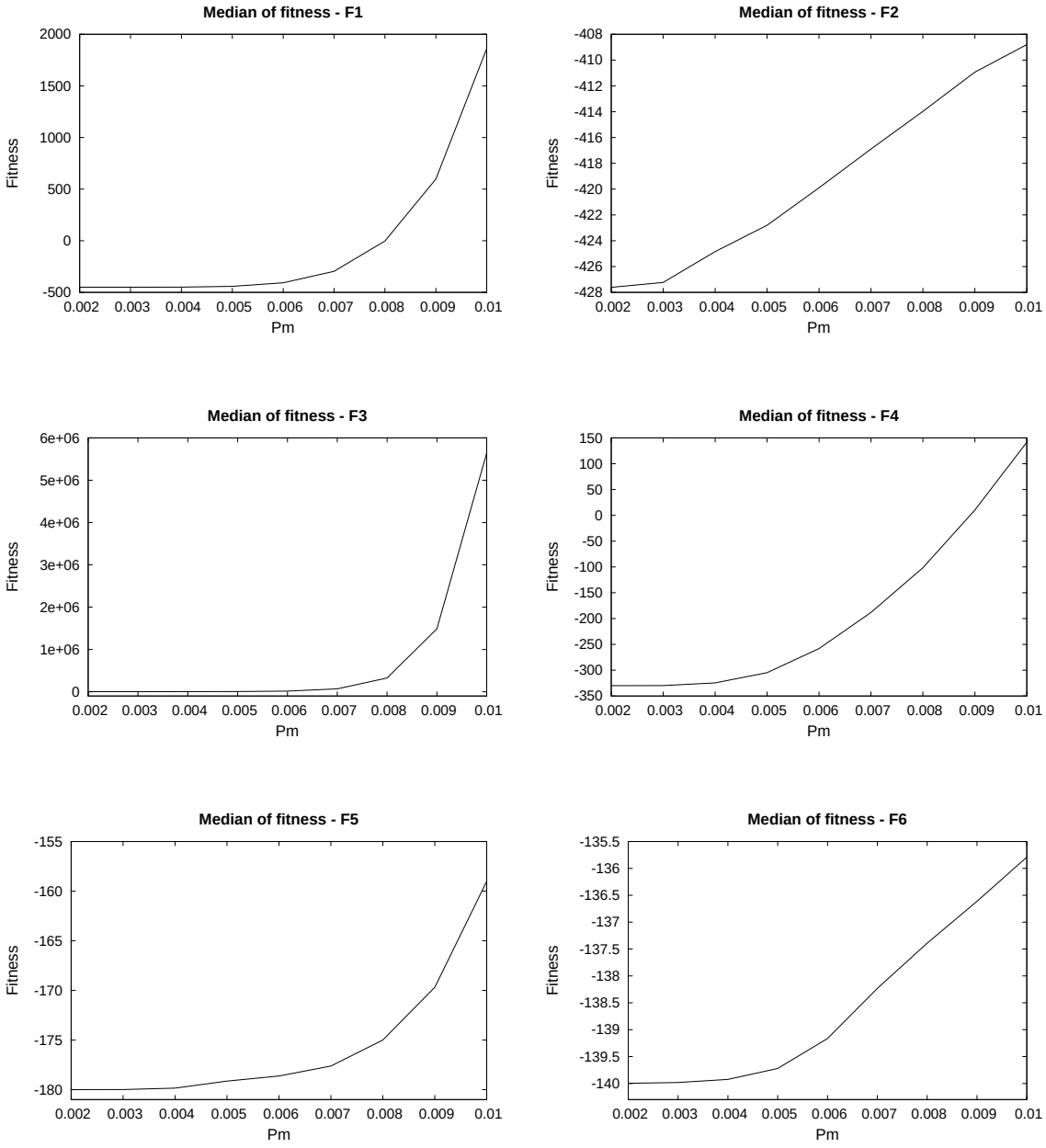


Figure 5.1: Median of the fitness considering different mutation probabilities (F1 - F6)

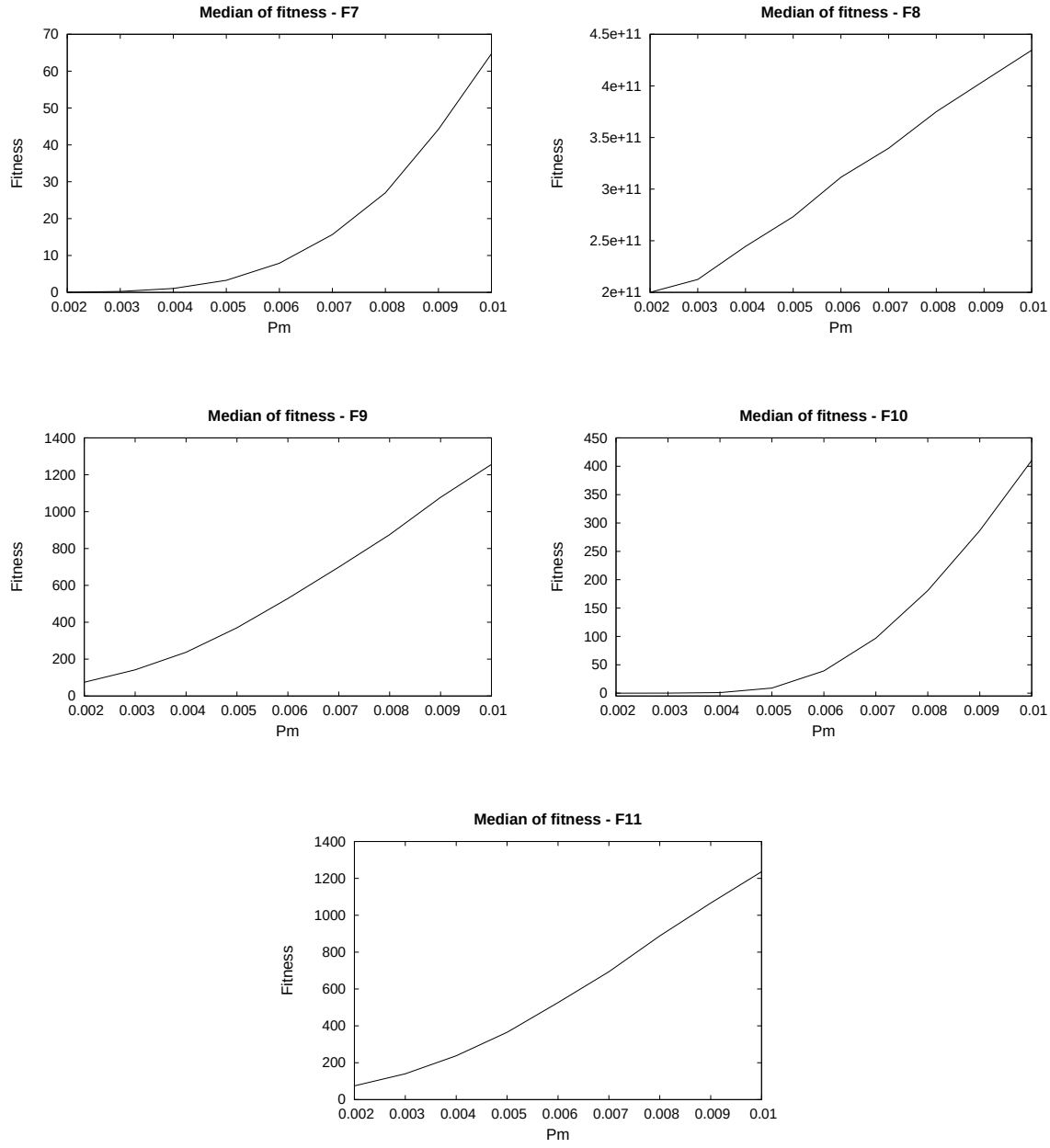


Figure 5.2: Median of the fitness considering different mutation probabilities (F7 - F11)

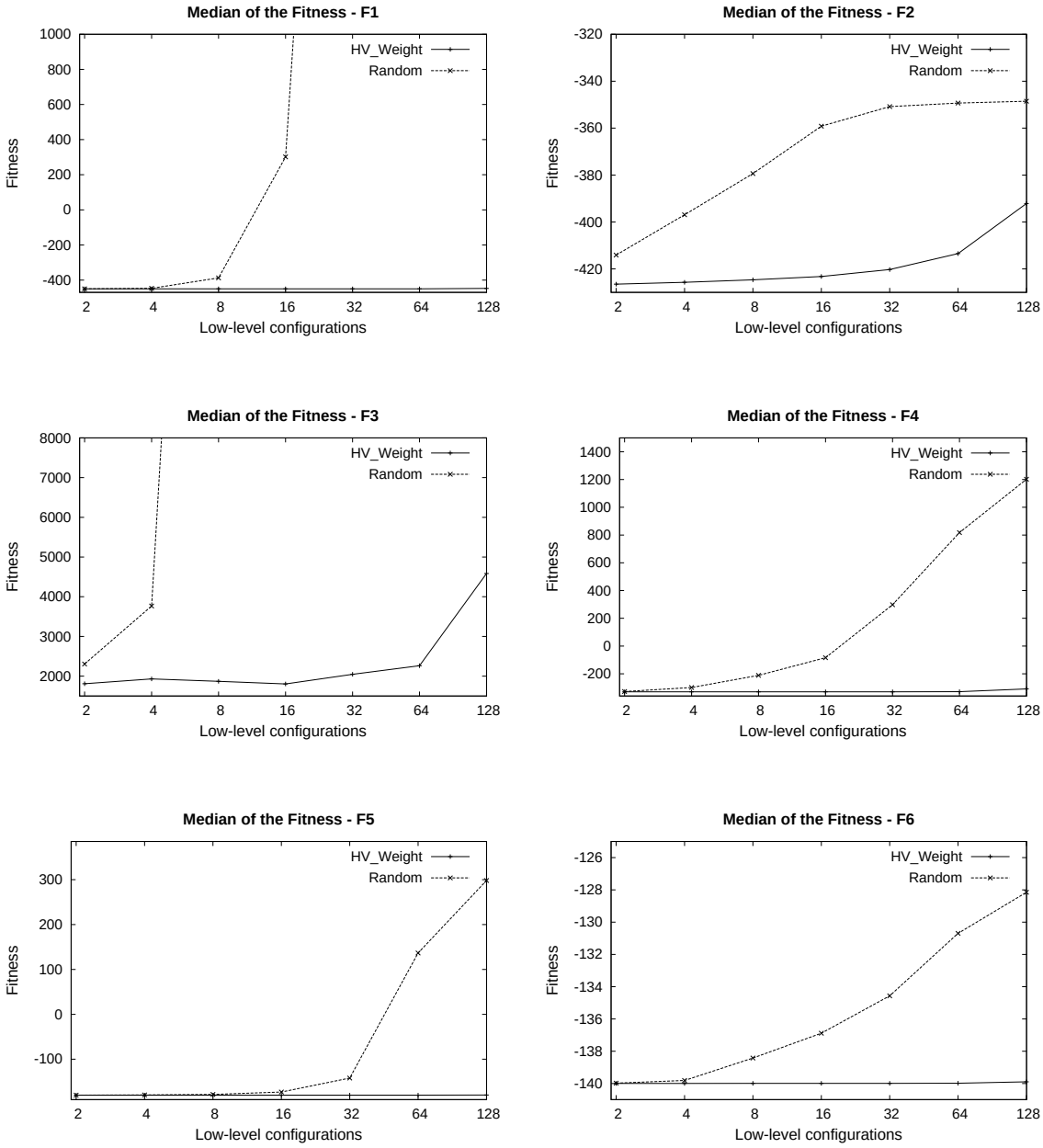


Figure 5.3: Median of the fitness considering different number of configurations (F1 - F6)

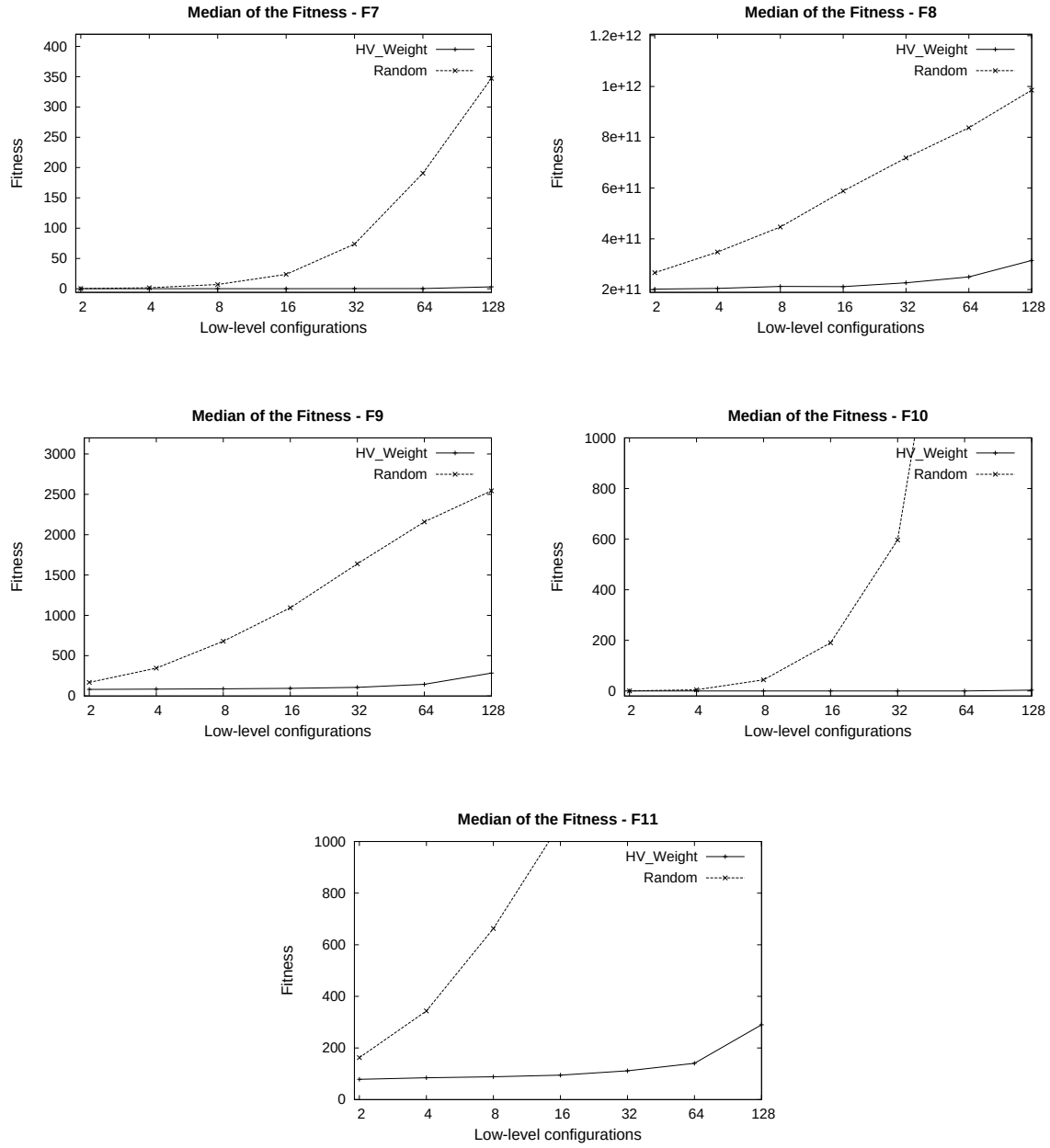


Figure 5.4: Median of the fitness considering different number of configurations (F7 - F11)

Table 5.2: Percentage of saved evaluations with MONO-WEIGHT (F1 - F6)

	F1	F2	F3	F4	F5	F6
2 *conf.*	46.00%	45.60%	46.34%	47.60%	46.93%	44.57%
4 *conf.*	69.60%	67.20%	63.45%	70.56%	69.35%	69.51%
8 *conf.*	80.80%	78.00%	77.91%	80.80%	80.97%	81.78%
16 *conf.*	81.52%	79.20%	76.75%	81.60%	82.00%	85.36%
32 *conf.*	76.85%	74.48%	73.06%	81.48%	76.80%	83.06%
64 *conf.*	69.60%	65.72%	69.91%	70.00%	71.20%	71.36%
128 *conf.*	32.82%	44.00%	60.24%	42.55%	45.08%	38.09%

Table 5.3: Percentage of saved evaluations with MONO-WEIGHT (F7 - F11)

	F7	F8	F9	F10	F11
2 *conf.*	48.19%	44.35%	45.20%	46.55%	44.97%
4 *conf.*	67.46%	68.69%	70.00%	69.47%	68.00%
8 *conf.*	74.80%	79.09%	79.43%	78.00%	79.67%
16 *conf.*	73.38%	82.18%	83.13%	81.52%	81.89%
32 *conf.*	65.44%	81.85%	80.75%	77.91%	81.25%
64 *conf.*	56.61%	76.82%	70.95%	69.13%	71.60%
128 *conf.*	23.70%	70.80%	48.00%	42.35%	47.79%

of 2.500.000 function evaluations. Figures 5.3 and 5.4 show, for each benchmark problem, the median of the fitness achieved at the end of the executions. Results are shown both for the MONO_WEIGHT and the RANDOM selection methods. Tests with up to 128 low-level configurations have been carried out. The advantages of using MONO_WEIGHT are clear. Thus, assigning the resources using the designed methodology is better than assigning them in a random way.

In addition, it is important to remark that in most of the cases as the number of configuration increases, the obtained fitness gets worse. The reason is that when many configurations are used it is more difficult to discriminate among suitable and non-suitable configurations. Moreover, in such cases a larger amount of evaluations are used in the initial stage where every low-level approach is executed. In addition, due to the way in which the low-level configurations have been defined, in every scheme there is at least one adequate low-level approach. Therefore, the obtained quality decreases as more configurations are added. It is important to remark that this might not happen in every case. For instance, a model with few configurations

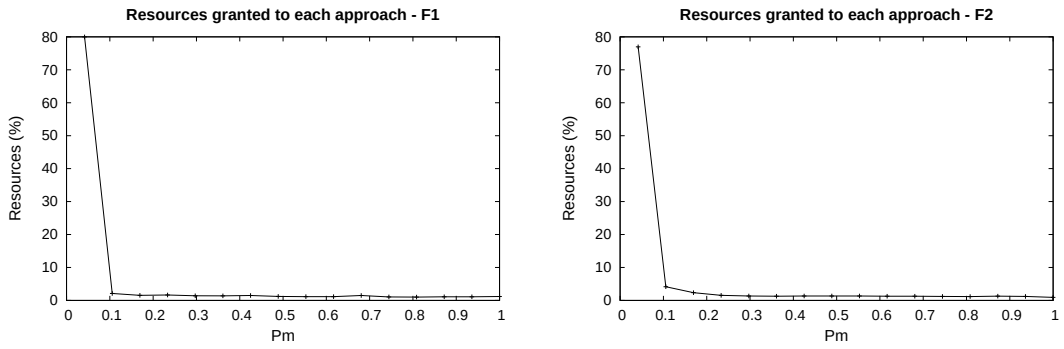


Figure 5.5: Resource sharing with 16 configurations (F1 - F2)

in which none of them are adequate, will behave worse than a model with many configurations, in which some of them are suitable for the problem. For this reason, the addition of alternative configurations might increase or decrease the quality of the obtained results. Anyway, even in the cases in which many configurations have been used, the MONO_WEIGHT model has obtained better results than the RANDOM approach. In fact, the statistical analyses have confirmed that the solutions obtained with MONO_WEIGHT have been better than the ones obtained with the RANDOM approach in every case.

The previous analysis has demonstrated some of the advantages of MONO_WEIGHT. However, it is also important to quantify the improvement. Tables 5.2 and 5.3 show the percentage of evaluations that can be saved by using the MONO_WEIGHT model instead of the RANDOM approach. It has been calculated considering the number of evaluations required to achieving a 50% of success ratio with each one of the models. The desired fitness has been calculated in the following way. First, the median of the fitness obtained by the MONO_WEIGHT and RANDOM models at the end of the executions have been calculated. The lowest value has been considered as the desired quality level. This ensures that both models attain the desired quality level. The saved percentages show that by using the MONO_WEIGHT approach, a large amount of resources can be saved. In fact, in some cases, more than 80% of resources can be saved. The reason of obtaining so large percentages is that some of the configurations are completely disruptive, so by avoiding them, a large quantity of resources can be saved. Moreover, such configurations might be used in some stages with the aim of avoiding premature convergence, improving on the obtained results. When many configurations have been used, the saved evaluations have not been so large. The reason is that in such cases the initial stage where

Table 5.4: Factor of additional evaluations performed with MONO-WEIGHT (F1 - F6)

	F1	F2	F3	F4	F5	F6
2 *conf.*	1.05	1.07	1.16	1.05	0.66	1.09
4 *conf.*	1.09	1.11	1.42	1.10	0.66	1.14
8 *conf.*	1.15	1.17	1.29	1.13	0.73	1.18
16 *conf.*	1.23	1.26	1.16	1.24	0.78	1.27
32 *conf.*	1.34	1.46	1.69	1.41	0.89	1.39
64 *conf.*	1.77	1.94	2.11	1.81	1.81	1.77
128 *conf.*	3.95	4.31	4.22	3.47	3.73	3.45

Table 5.5: Factor of additional evaluations performed with MONO-WEIGHT (F7 - F11)

	F7	F8	F9	F10	F11
2 *conf.*	1.06	1.02	1.08	1.05	1.03
4 *conf.*	1.14	1.04	1.11	1.09	1.09
8 *conf.*	1.24	1.13	1.14	1.20	1.13
16 *conf.*	1.40	1.11	1.21	1.22	1.20
32 *conf.*	1.60	1.33	1.33	1.34	1.38
64 *conf.*	2.02	1.71	1.73	1.70	1.71
128 *conf.*	5.32	3.01	3.20	3.73	3.27

each configuration is executed one time is larger, so more resources are granted to non-adequate strategies.

The MONO_WEIGHT approach has obtained better results than the RANDOM scheme in terms of the obtained fitness. This indicates that more resources are granted to the most suitable low-level approaches in every case. The sharing of resources has been measured for the model that considers 16 low-level configurations. Figure 5.5 show the percentage of resources granted to each low-level configuration for the problems F1 and F2. The sharing for the rest of the problems is not shown because it was similar in every case. It can be appreciated that a higher amount of resources are granted to the low-disruptive configurations. This means that the hyperheuristic is detecting which are the most suitable configurations, and more resources are granted to them.

The increase on the generality of the hyperheuristics is usually achieved at the

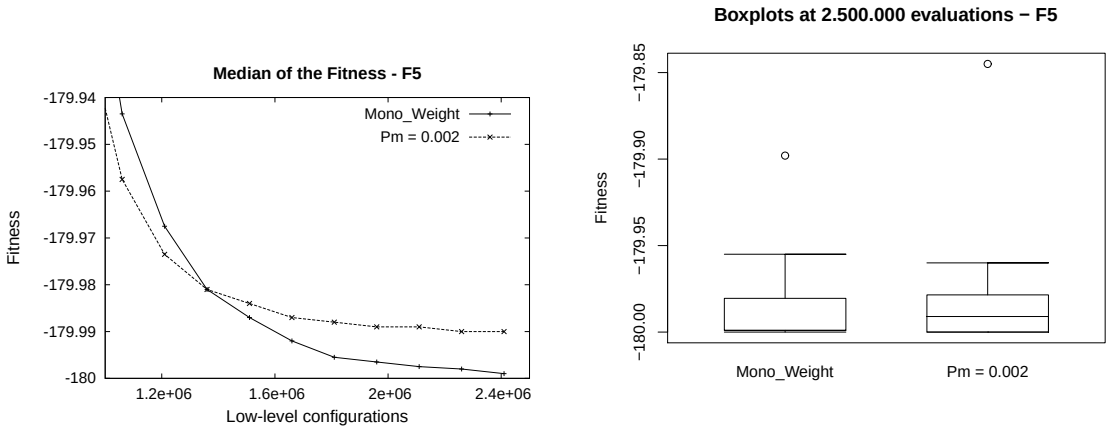


Figure 5.6: Fitness for F5 with different configurations

expense of reduced solution quality when compared to the best low-level approaches. For this reason, it is also important to quantify the relative performance with respect to the best low-level configuration. In order to give a measure of the impact produced by the hyperheuristics, the factors between the evaluations required to obtain a 50% of success ratio, with the MONO_WEIGHT model and with the best sequential low-level approach have been calculated. The desired quality level was fixed as in the previous experiment. Tables 5.4 and 5.5 show such a value for each problem and for the different number of considered configurations. It can be observed that, as it was expected, in most cases the hyperheuristic requires a larger amount of evaluations than the corresponding best low-level approach. Moreover, the factors increase as more configurations are considered. However, in every case the factor is much lower than the number of low-level configurations. Since the testing of each low-level configuration can be avoided by using the hyperheuristic, the overall computing and user effort saved is very large.

In the theoretical analysis, it was shown that in some cases, the hyperheuristics might obtain better values than any of the involved low-level approaches. The previous experimental analysis shows that MONO_WEIGHT required fewer evaluations than the best sequential approach in F5, demonstrating that, in practice, it also happens. In such a case the hyperheuristic has been better than the best low-level approach, when less than 32 configurations have been used. In the case of using two low-level configurations the number of evaluations required by the hyperheuristic has been much lower than the evaluations required by the best low-level approach. Figure 5.6 shows the evolution of the median of the fitness for the best low-level approach, and

for MONO_WEIGHT. The boxplots at the end of the executions are also shown. It can be appreciated that the hyperheuristic attains better values. In fact, most of the executions are near to the optimal value. Therefore, the synergy of combining a low-disruptive configuration with a high-disruptive configuration is clear. The statistical tests do not confirm the superiority of the hyperheuristic with a stopping criterion of 2.500.000 function evaluations. However, in the case of using as stopping criterion the execution of 4.000.000 function evaluations, the statistical tests confirm the superiority of MONO_WEIGHT.

Second Experiment: Adaptation level

Previous analyses have shown the advantages of using MONO_WEIGHT with the parameterisation $k = 5$. It is interesting to analyse the robustness of the approach with respect to the value of k. In order to check it, the model MONO_WEIGHT was executed with the F tests, but considering the parameterisations $k = \{1, 5, 10, 100, 1000, \infty\}$. As it was described in the theoretical analysis, intermediate values (100 and 1000) are not expected to obtain high-quality solutions. Anyway, it is useful to test them with the aim of confirming it. In every case, 32 low-level configurations were considered. They were made up as in the previous experiment. The remaining parameters of the hyperheuristic were also the same than in the previous experiment. The medians of the fitness obtained at the end of the executions are shown in tables 5.6 and 5.7. The model MONO_WEIGHT with low-vales of k has reported much better results than an assignment based on the RANDOM approach in every case. Other values of k have also reported better values than the RANDOM approach in most of the cases. However, in F7, high values of k are not adequate. The reason is that the hyperheuristic get confounded by the results obtained in previous stages of the optimisation. Also, as it was expected, intermediate values of k are, in general, not adequate when compared to low and high values of k. The parameterisation $k = 1$ has been the one that has reported the best values in every case. Moreover, in most cases, differences with the results obtained by using other values of k have been statistically significant. Therefore, with such low-level approaches estimating the score by using the average of several executions is not required. The reasons are that differences among the configurations are very large, and that a large stopping criterion has been used. Therefore, the last obtained improvement is a good score, and there is no need of using a larger value of k.

The model was also executed with other low-level approaches. In such cases, the previous behaviour could not be reproduced. In the additional experiments, several cases were found in which the value $k = 1$ was not the best one. A case in which this happened is presented in the following lines.

Table 5.6: Median of the fitness for different values of k (F1 - F6)

	F1	F2	F3	F4	F5	F6
$k = 1$	-449.98	-420.93	1978.65	-329.98	-179.99	-139.99
$k = 5$	-449.98	-420.27	2046.16	-329.96	-179.99	-139.99
$k = 10$	-449.97	-417.53	2155.19	-329.84	-179.98	-139.99
$k = 100$	-447.70	-411.34	13730.15	-319.79	-179.80	-139.98
$k = 1000$	-447.72	-410.69	11794.8	-323.82	-179.58	-139.98
$k = \infty$	-449.97	-416.67	2661.7	-329.96	-179.99	-139.99
Random	6101.32	-350.83	48584650	297.99	-141.95	-134.56

Table 5.7: Median of the fitness for different values of k (F7 - F11)

	F7	F8	F9	F10	F11
$k = 1$	0.10	2.19e11	100.06	0.008	98.83
$k = 5$	0.15	2.26e11	107.67	0.011	111.54
$k = 10$	0.20	2.36e11	120.91	0.019	124.23
$k = 100$	587.49	3.44e11	179.82	3.42	174.24
$k = 1000$	625.78	3.56e11	157.69	3.06	170.42
$k = \infty$	539.80	2.50e11	117.89	0.01	120.52
Random	73.72	7.18e11	1638.98	596.61	1615.18

The model MONO_WEIGHT was executed by considering 18 low-level configurations. They were made up by combining three survivor selection schemes, three crossover operators and two mutation operators. The three tested survivor selection schemes were: SS-S, GEN-S, and RW-S. The applied crossover operators were: UX, OPX and SBX. Finally, the tested mutation operators were the UM and the PM. The parameter $\beta * n_h$ was set to 0.1, while the following values of k were tested: 1, 5, 10, 100, 1000, ∞. The local stopping criterion was set to 1.000 function evaluations. The RANDOM approach was also executed. In every case, the global stopping criterion was set to 10.000.000 function evaluations.

In most of the F problems, the model reported results that were very similar to the ones obtained by best sequential approach. The best-behaved value of k depended on the considered problem. Figure 5.7 shows, for the F11 problem, the evolution of the median of the fitness attained with MONO_WEIGHT with the different values of k. It also shows the evolution of the RANDOM approach. Initially, the value $k = 1$ is the most adequate. However, in the long term, the MONO_WEIGHT approaches with

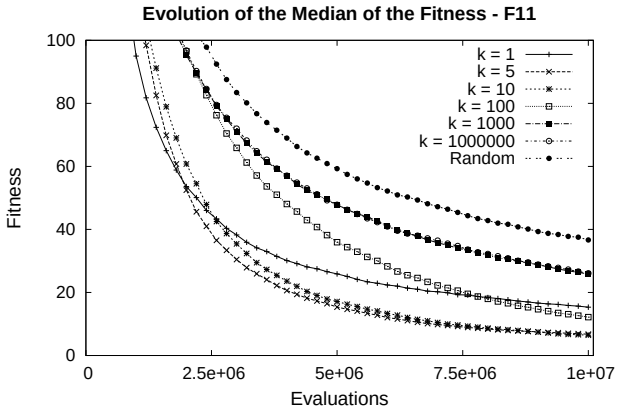


Figure 5.7: Evolution of the fitness with MONO_WEIGHT for different adaptation levels

higher values of k (5 and 10) clearly improve on the results obtained with $k = 1$. In fact, the model MONO_WEIGHT with $k = 5$ and $k = 10$ has reported statistically better values than the model with $k = 1$. This confirms the requirement of considering the average of the last improvements (and not only the last improvement) in some problems. The main inconvenient is that the best-behaved value of k depends on the problem. The main strength is that the results obtained with low values of k have improved on the results obtained with the RANDOM scheme in every case.

Third Experiment: Improving on the results obtained with the best low-level approach

During the development of the second experiment, several cases were found in which the hyperheuristic was statistically better than the best considered low-level approach. In the first experiment it had already happened with F5. In such a case the cause of the good performance was that by combining low-disruptive schemes, with high-disruptive schemes, premature convergence was avoided. However, in the new detected cases premature convergence was not appearing, so the reasons of the good performance are analysed in this experiment.
One of the models in which MONO_WEIGHT improved on the results obtained by the best sequential approach was the one exposed in the second experiment. Figure 5.8 shows the boxplots of the fitness obtained at the end of the executions by the best sequential approach and by MONO_WEIGHT with $k = 5$. The advantages of using the hyperheuristic are clear.

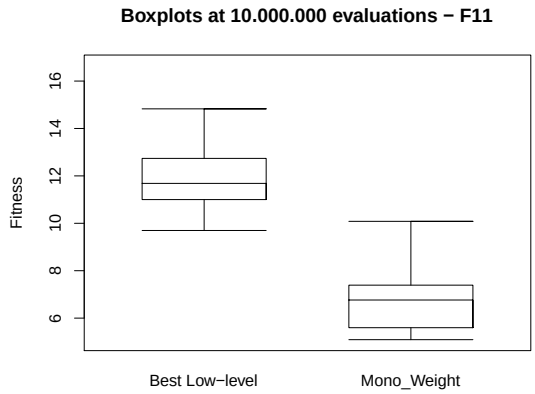


Figure 5.8: Comparison of the best sequential approach and MONO_WEIGHT

In order to deeply analyse the reason of the good behaviour of MONO_WEIGHT, the behaviour of the low-level approaches was analysed for the different stages of the optimisation. In the first stage of MONO_WEIGHT, every configuration is executed one time. After it, the median of the fitness value has been about 800. At this initial stage, the hyperheuristic does not perform any decision. In the following stages, the hyperheuristic performs its decisions based on the quality of the low-level approaches. Thus, an analysis of the behaviour of the sequential low-level approaches in the subsequent stages has been performed.

Table 5.8 shows for each of the low-level sequential configurations the median of the fitness achieved when they start from a population with individuals whose fitness values are close to 800. Executions have been performed taking into consideration a stopping criterion of 10^5 evaluations. Figure 5.9 shows the boxplots of the fitness achieved by the best configurations. It shows the similarities between the best six configurations. In fact, differences among them are not statistically significant.

After reaching a fitness value close to 46, the behaviour of the configurations is completely different. Table 5.9 shows that, when starting from a population with individuals whose fitness values are close to 46, some of the configurations which behaved poorly in the previous stages are now the best-behaved ones. The shown fitness values have been obtained taking into consideration a stopping criterion of 10^5 evaluations. Figure 5.10 shows the boxplots of the fitness achieved by the seven best configurations. In this case, differences among the best configuration and the remaining ones are statistically significant.

Table 5.8: Median of the fitness achieved in 10^5 evaluations with F11 (starting from 800)

Name	Mutation	Crossover	Selection	Achieved Fitness
Seq1	PM	OPX	SS-S	336.53
Seq2	PM	SBX	SS-S	338.76
Seq3	PM	SBX	RW-S	339.59
Seq4	PM	OPX	RW-S	339.94
Seq5	PM	UX	RW-S	340.56
Seq6	PM	UX	SS-S	344.60
Seq7	PM	SBX	GEN-S	404.14
Seq8	PM	UX	GEN-S	439.76
Seq9	PM	OPX	GEN-S	443.69
Seq10	UM	SBX	GEN-S	580.06
Seq11	UM	OPX	RW-S	615.17
Seq12	UM	OPX	SS-S	623.82
Seq13	UM	SBX	RW-S	635.90
Seq14	UM	SBX	SS-S	638.70
Seq15	UM	UX	RW-S	643.57
Seq16	UM	UX	SS-S	647.16
Seq17	UM	OPX	GEN-S	692.88
Seq18	UM	UX	GEN-S	697.94

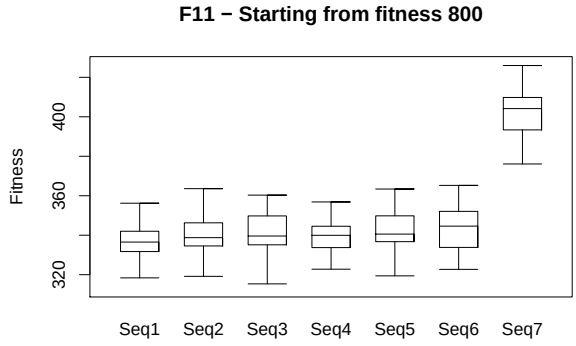


Figure 5.9: Boxplots of the fitness achieved in 10^5 evaluations with F11 (starting from 800)

Table 5.9: Median of the fitness achieved in 10^5 evaluations with F11 (starting from 46)

Name	Mutation	Crossover	Selection	Achieved Fitness
Seq1	PM	SBX	GEN-S	42.83
Seq2	PM	OPX	RW-S	43.76
Seq3	PM	OPX	SS-S	43.89
Seq4	UM	OPX	RW-S	44.11
Seq5	UM	SBX	GEN-S	44.41
Seq6	UM	OPX	SS-S	44.50
Seq7	UM	OPX	GEN-S	44.91
Seq8	PM	OPX	GEN-S	45.03
Seq9	PM	UX	RW-S	45.48
Seq10	PM	UX	SS-S	45.49
Seq11	PM	SBX	RW-S	45.51
Seq12	PM	SBX	SS-S	45.53
Seq13	PM	UX	GEN-S	45.71
Seq14	UM	SBX	SS-S	45.86
Seq15	UM	UX	SS-S	45.87
Seq16	UM	UX	RW-S	45.87
Seq17	UM	SBX	RW-S	45.87
Seq18	UM	UX	GEN-S	45.87

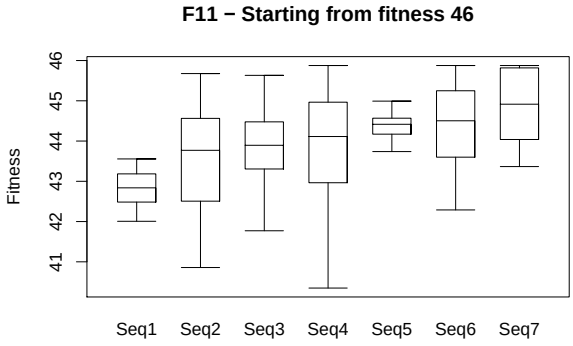


Figure 5.10: Boxplots of the fitness achieved in 10^5 evaluations with F11 (starting from 46)

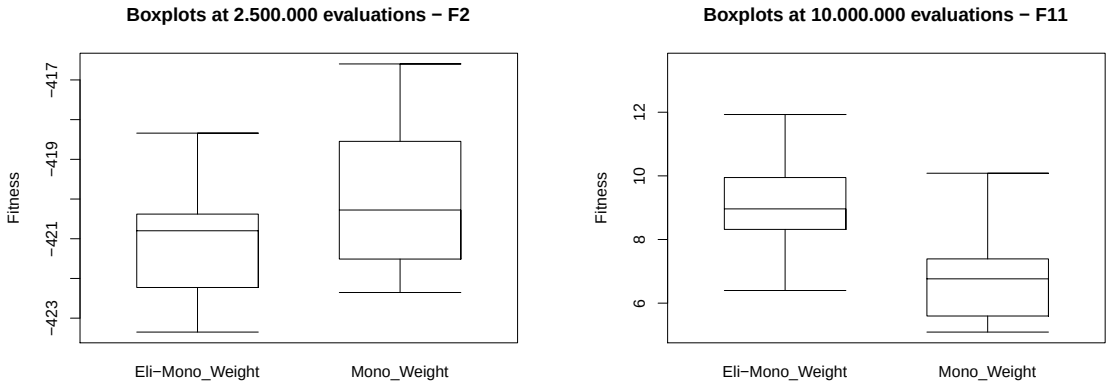


Figure 5.11: Comparison of elitist and probabilistic selection schemes

Therefore, the relative performance among the low-level approaches depends on the stage of the optimisation. The MONO_WEIGHT model has been able to grant more computational resources to the best approach in each stage of the optimisation. Therefore, it has been able to improve on the results obtained by every low-level approach.

Fourth Experiment: Probabilistic vs. Elitist Selection

An experiment with the aim of comparing the probabilistic and elitist selection schemes was also performed. Several configurations of the ELI-MONO_WEIGHT and MONO_WEIGHT hyperheuristics were executed. In most of the cases, the executions that used ELI-MONO_WEIGHT obtained slightly better values than the ones that used MONO_WEIGHT. The reason is that MONO_WEIGHT tends to distribute the resources among more configurations, while ELI-MONO_WEIGHT tends to focus on few configurations. In most cases, such type of assignment clearly provides advantages. However, in other cases, the hyperheuristic is confounded and assigns a high number of resources among non-suitable configurations. In such cases, the loss of performance is large. Two opposite cases are presented. In the first one the hyperheuristic was configured as in the first experiment. Specifically, 32 configurations were used, and it was applied to the F2 problem. Figure 5.11 shows - in the left - the boxplots of the results obtained by ELI-MONO_WEIGHT and by MONO_WEIGHT. It shows that by using an elitist selection scheme, better fitness values can be obtained. In addition, executions that considered the low-level configurations applied

in the second experiment were also carried out. In this case it was applied to the F11 problem. Figure 5.11 shows - in the right - the boxplots of the results obtained by ELI-MONO_WEIGHT and by MONO_WEIGHT. The probabilistic selection scheme obtains better results. Thus, the most suitable selection scheme depends on the problem and low-level approaches. Moreover, in both cases, differences have been significant. Anyway, in both cases good quality values have been obtained compared with the RANDOM approach. Therefore, by properly specifying the kind of selection better values can be obtained, but independently of the kind of selection high quality results have been obtained.

Fifth Experiment: Performance of the parallel hyperheuristic

This section is devoted to present the experiments that have been performed with the aim of exploring the performance of the parallel version of the hyperheuristics. Specifically, a hyperheuristic that considered 16 configurations - the same as in the first experiment - was executed in parallel. The same parameterisation as in the case of the first experiment was used. Specifically, the value of k was fixed to 5, and the value of β was fixed in a way that the 10% of the decisions performed by the hyperheuristic followed a uniform distribution, i.e., $\beta * n_h = 0.1$. Finally, the local stopping criterion was fixed to 10.000 function evaluations and the global stopping criterion to 2.500.000 function evaluations. Thus, the only change was the specification of the number of processors to use.
Since it is known that the migration stage might affect the performance of island-based models, the aim of the first analysis has been to measure the impact that the migration stage has over the performance. The hyperheuristic was executed with 4 processors considering several types of migration stages. Specifically, 12 different migrations stages were considered. They were made up by combining two migration selectors, three exchange selectors and two topologies. The migration selectors were the ELITIST, the IMPROVEMENTS and the RANDOM. The ELITIST approach selects the best individuals of the population. The IMPROVEMENTS approach selects individuals that are better than any individual of its previous population. If such an individual does not exist, no migration is performed. The RANDOM selector picks up the individuals randomly. The considered exchange selectors were the RANDOM, and the ELITIST. The RANDOM approach selects the individual to exchange randomly. The ELITIST approach discards the worst individuals among the current population and the immigrants. Finally, the tested topologies were the ALL and the RING. Some preliminary tests were performed with the aim of setting the rest of the parameters of the migration stage. The number of individuals to migrate was fixed to 1, while the probability of migration was fixed to 100%, i.e. after every

Table 5.10: Speedup of the parallel approach (4 islands) with MONO-WEIGHT and different migration stages (F1 - F6)

	F1	F2	F3	F4	F5	F6
ELITIST-ELITIST-ALL_ALL	2.60	3.82	2.55	3.28	2.71	3.70
IMPROVEMENTS-ELITIST-ALL_ALL	3.20	3.66	3.14	3.71	3.26	3.77
RANDOM-ELITIST-ALL_ALL	3.11	3.50	3.14	3.57	3.16	4.05
ELITIST-RANDOM-ALL_ALL	2.73	3.96	2.90	3.42	2.74	3.85
IMPROVEMENTS-RANDOM-ALL_ALL	3.38	3.71	2.95	3.85	3.43	3.83
RANDOM-RANDOM-ALL_ALL	0.17	0.04	0.41	0.11	0.13	0.12
ELITIST-ELITIST-RING	3.01	3.66	3.01	3.61	3.11	3.70
IMPROVEMENTS-ELITIST-RING	3.30	3.63	3.01	3.76	3.49	3.70
RANDOM-ELITIST-RING	3.41	3.63	3.01	3.71	3.11	3.66
ELITIST-RANDOM-RING	3.12	3.73	3.07	3.70	3.37	3.77
IMPROVEMENTS-RANDOM-RING	3.30	3.60	3.18	3.72	3.53	3.54
RANDOM-RANDOM-RING	0.39	0.81	0.51	0.38	0.39	0.33

Table 5.11: Speedup of the parallel approach (4 islands) with MONO-WEIGHT and different migration stages (F7 - F11)

	F7	F8	F9	F10	F11
ELITIST-ELITIST-ALL_ALL	2.71	2.87	2.84	2.66	2.85
IMPROVEMENTS-ELITIST-ALL_ALL	3.07	3.19	2.90	2.80	2.98
RANDOM-ELITIST-ALL_ALL	3.01	2.87	3.00	3.11	3.04
ELITIST-RANDOM-ALL_ALL	2.85	2.85	2.90	2.79	2.84
IMPROVEMENTS-RANDOM-ALL_ALL	3.20	3.17	3.36	3.11	3.11
RANDOM-RANDOM-ALL_ALL	0.71	0.34	0.14	0.14	0.11
ELITIST-ELITIST-RING	3.03	2.98	3.14	3.04	3.12
IMPROVEMENTS-ELITIST-RING	3.17	2.95	3.15	3.15	3.22
RANDOM-ELITIST-RING	3.72	3.17	3.22	3.46	3.14
ELITIST-RANDOM-RING	3.27	3.09	3.15	3.11	3.19
IMPROVEMENTS-RANDOM-RING	3.42	3.24	3.28	3.23	3.25
RANDOM-RANDOM-RING	0.73	0.48	0.33	0.38	0.35

generation the migration stage was executed.
In order to measure the performance of the parallel approach, the sequential version of MONO_WEIGHT that considered 16 configurations was used to perform the com-

parison. Tables 5.10 and 5.11 show the speedup factors obtained with the different models for each one of the F problems. The factors have been calculated using the evaluations required by each model to achieving a 50% of success ratio. The quality level has been fixed in a similar way than in previous experiments. The calculated speedup factor assumes that in the parallel approaches the number of evaluations is n_p times higher than the evaluations carried out by the sequential approaches in the same time. This is the typical behaviour when island-based models are applied to real-world problems. Thus, considering that $SeqEval$ and $ParEval$ are the number of evaluations required to obtaining the desired success ratio for a sequential and a parallel approach, the speedup factor has been calculated as:

$$speedup = \frac{SeqEval}{ParEval} \cdot n_p \tag{5.1}$$

The calculated speedup factors reveal that the migration stage has some effect over the speedup. The schemes that use a random migration selector with a random exchange selector were not able to accelerate the achievement of high quality solutions. The reason is that they might replace the best individuals with low-quality individuals. The rest of the migration stages obtained acceptable speedup factors. The migration stages IMPROVEMENTS-RANDOM-RING and IMPROVEMENTS-ELITIST-RING obtained high speedup factors in every case. In addition, the amount of data sent with such an approach over the network is lower than with the other migration stages. Thus, they have been selected to perform a scalability analysis.
First, such models were executed with the same parameterisation than in the previous experiment, but considering up to 32 islands. The models did not scale. The reason was that since several islands get idle at the same time, the hyperheuristic tends to use the same approach in many islands. When the selection is not adequate a lot of resources are wasted. For this reason, the local stopping criterion was reduced to 500 function evaluations. In this case, better speedup factor were obtained. In addition some experiments that considered the ALL topology and up to 32 islands were also executed. They saturated the network, so the ALL topology was not scalable with problems with so many variables. Such a topology was also used with lower migration probabilities. The calculated speedup factor revealed that such a setting was not adequate.
Figures 5.12 and 5.13 show the speedup factors obtained with the selected migration stages, and with the stopping criterion fixed to 500 function evaluations. A linear speedup could not be obtained. The reason is that since the global population is much larger than in the sequential case, the way in which the fitness landscape is

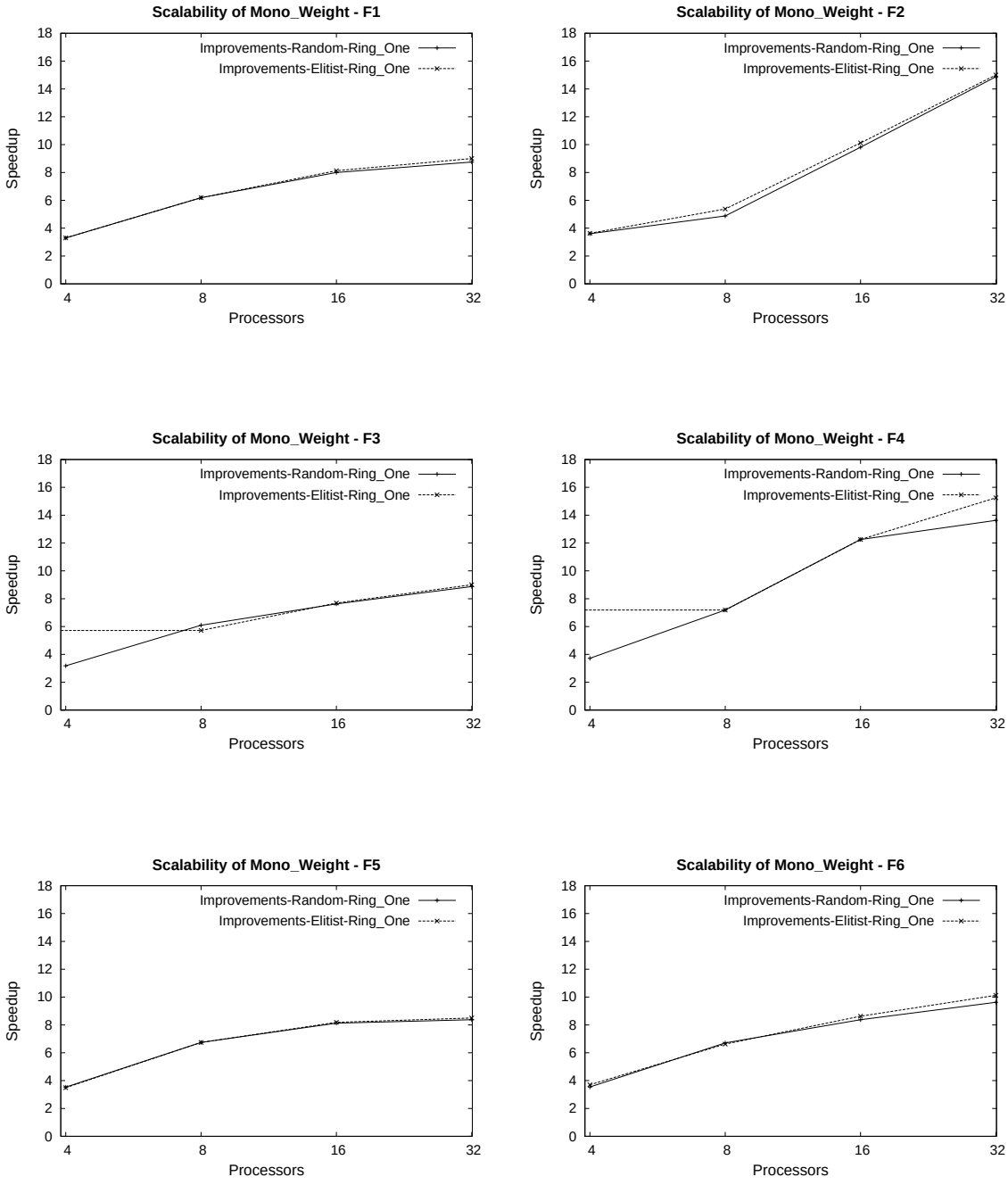


Figure 5.12: Scalability analysis for the best migration stages (F1 - F6)

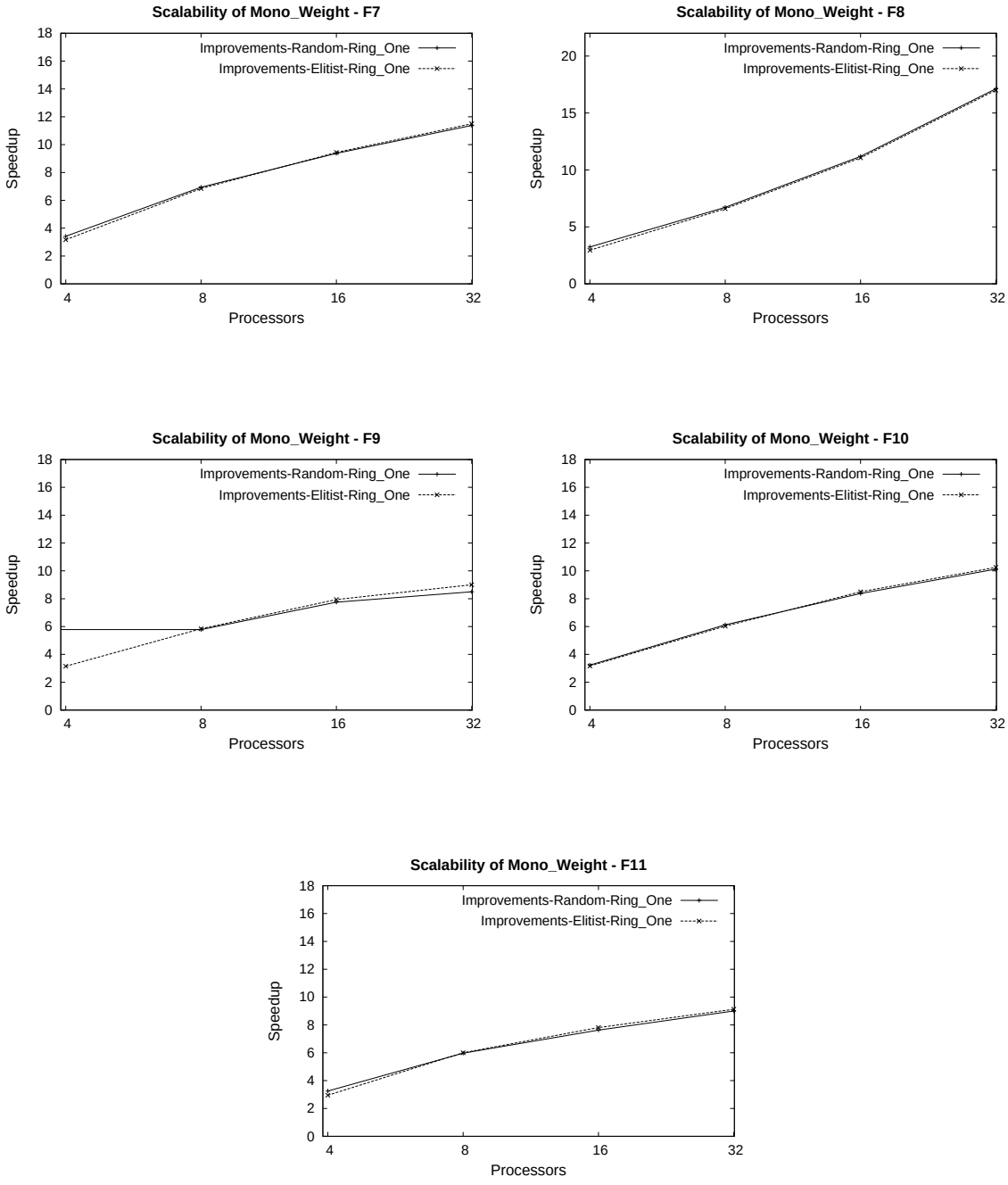


Figure 5.13: Scalability analysis for the best migration stages (F7 - F11)

explored is very different. In fact, with the performed migration stage, the model is similar to a sequential scheme that creates in each generation an offspring set much larger than the population. Anyway, in any of the cases as more processors have been added, the speedup factors have increased. The most outstanding case with 32 processors appeared with F8. In such a case the speedup factor was 17.25. The problem that produced the worst case with 32 processors was F5. The obtained speedup factor was 8.37. Thus, with few processors it is very easy to obtain adequate speedup factors. However, for larger number of processors the migration stage must be carefully selected, and the obtained speedup is not linear.

5.1.3 Experimental Evaluation of Schemes Based on Multiobjectivisation

This section shows the experimental evaluation performed with the schemes based on multiobjectivisation. First, the advantages of using the multiobjectivisation schemes based on Euclidean distances are explored. Then, an analysis of the benefits of including a threshold for the fitness in the multiobjectivised approaches is performed. Specifically, the performance of the new DCN-THR scheme is analysed. Finally, the adaptive multiobjectivisation is evaluated.

First Experiment: On the advantages of multiobjectivisation

The first analysis has been devoted to measure the performance of multiobjectivisation. A comparison between mono-objective and multiobjectivised approaches has been performed. In the case of the mono-objective optimisation, the basic EAs presented in Chapter 2 has been used. Three different survivor selection mechanisms have been considered: SS-S, GEN-S, and RW-S. For the multiobjectivised approaches, NSGA-II has been applied. The multiobjectivisation has been performed with three different mechanisms: DCN, ADI, and DBI. In both cases, the variation stage has been based on using the UM with $p_m = \frac{1}{D}$, and the SBX with $p_c = 1$. The F problems have been tested with 50 and 500 variables (D). Each algorithm has been executed with a population size N of 5, 10, and 20 individuals. Finally, the stopping criterion has been fixed to a total number of $5000 \cdot D$ evaluations.
Table 5.12 shows, for each population size, the best mono-objective and multiobjectivised approaches considering $D = 50$. Comparisons have been made in terms of the median of the fitness achieved at the end of each execution. The same information is shown in Table 5.14 considering $D = 500$. In both cases, the superiority of the *GEN-S* and the DCN strategies is clear. However, for some problems, they are not the best-behaved schemes.

Table 5.12: Best mono-objective and multi-objective approaches - $D = 50$

	F1	F2	F3	F4	F5	F6
Mono_5	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Mono_10	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Mono_20	GEN-S	GEN-S	GEN-S	RW-EA	GEN-S	GEN-S
Multi_5	DCN	DCN	DCN	DCN	DCN	DCN
Multi_10	DCN	DCN	DCN	DCN	DCN	DCN
Multi_20	DCN	DCN	DCN	DCN	DCN	DCN

	F7	F8	F9	F10	F11
Mono_5	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Mono_10	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Mono_20	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Multi_5	DCN	ADI	DCN	DCN	DCN
Multi_10	DCN	DCN	DCN	DCN	DCN
Multi_20	DCN	DCN	DCN	DCN	DCN

Table 5.13: Median of the error for the best approaches - $D = 50$

	F1	F2	F3	F4	F5	F6
Mono_5	$< 1 \cdot 10^{-6}$	1.76	155	$< 1 \cdot 10^{-6}$	$2.25 \cdot 10^{-2}$	$3.00 \cdot 10^{-3}$
Multi_5	$< 1 \cdot 10^{-6}$	**0.567**	155	$< 1 \cdot 10^{-6}$	$\mathbf{1.10 \cdot 10^{-2}}$	$\mathbf{1.00 \cdot 10^{-3}}$
Mono_10	$\mathbf{< 1 \cdot 10^{-6}}$	1.63	344	$< 1 \cdot 10^{-6}$	$1.50 \cdot 10^{-2}$	$\mathbf{2.00 \cdot 10^{-3}}$
Multi_10	$5.00 \cdot 10^{-4}$	**1.05**	**152**	$1.00 \cdot 10^{-3}$	$1.00 \cdot 10^{-2}$	$4.00 \cdot 10^{-3}$
Mono_20	$6.00 \cdot 10^{-3}$	2.01	338	$8.60 \cdot 10^{-2}$	$2.10 \cdot 10^{-2}$	$1.60 \cdot 10^{-2}$
Multi_20	$\mathbf{3.00 \cdot 10^{-3}}$	**1.64**	197	$\mathbf{1.00 \cdot 10^{-3}}$	$1.90 \cdot 10^{-2}$	$\mathbf{9.00 \cdot 10^{-3}}$

	F7	F8	F9	F10	F11
Mono_5	$4.57 \cdot 10^{-3}$	$1.44 \cdot 10^{9}$	7.24	$1.84 \cdot 10^{-4}$	6.80
Multi_5	$\mathbf{1.37 \cdot 10^{-3}}$	$\mathbf{4.50 \cdot 10^{8}}$	**4.65**	$\mathbf{3.11 \cdot 10^{-5}}$	**4.46**
Mono_10	$2.59 \cdot 10^{-3}$	$1.02 \cdot 10^{9}$	7.77	$\mathbf{1.16 \cdot 10^{-4}}$	7.05
Multi_10	$2.55 \cdot 10^{-3}$	$\mathbf{6.49 \cdot 10^{8}}$	**5.62**	$4.69 \cdot 10^{-4}$	**6.12**
Mono_20	$1.82 \cdot 10^{-2}$	$1.34 \cdot 10^{9}$	17.9	$5.40 \cdot 10^{-3}$	19.8
Multi_20	$\mathbf{3.70 \cdot 10^{-3}}$	$\mathbf{8.77 \cdot 10^{8}}$	**6.92**	$\mathbf{1.47 \cdot 10^{-3}}$	**8.00**

Table 5.14: Best mono-objective and multi-objective approaches - $D = 500$

	F1	F2	F3	F4	F5	F6
Mono_5	GEN-S	RW-EA	GEN-S	GEN-S	GEN-S	GEN-S
Mono_10	GEN-S	RW-EA	GEN-S	GEN-S	GEN-S	GEN-S
Mono_20	GEN-S	RW-EA	GEN-S	GEN-S	SS-EA	GEN-S
Multi_5	DCN	ADI	DCN	DCN	DCN	DCN
Multi_10	DCN	DCN	DCN	DCN	DCN	DCN
Multi_20	DCN	DCN	DCN	DCN	DCN	DCN

	F7	F8	F9	F10	F11
Mono_5	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Mono_10	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Mono_20	GEN-S	GEN-S	GEN-S	GEN-S	GEN-S
Multi_5	DCN	DCN	DCN	DCN	DCN
Multi_10	DCN	DCN	DCN	DCN	DCN
Multi_20	DCN	DCN	DCN	DCN	DCN

Table 5.15: Median of the error for the best approaches - $D = 500$

	F1	F2	F3	F4	F5	F6
Mono_5	$3.00 \cdot 10^{-3}$	16.7	$1.32 \cdot 10^{3}$	$3.00 \cdot 10^{-3}$	$8.00 \cdot 10^{-3}$	$3.00 \cdot 10^{-3}$
Multi_5	$\mathbf{< 1 \cdot 10^{-6}}$	**11.4**	$1.26 \cdot 10^{3}$	$3.00 \cdot 10^{-3}$	$< 1 \cdot 10^{-6}$	$\mathbf{1.00 \cdot 10^{-3}}$
Mono_10	$\mathbf{2.00 \cdot 10^{-3}}$	**14.1**	$1.47 \cdot 10^{3}$	$3.95 \cdot 10^{-2}$	$< 1 \cdot 10^{-6}$	$\mathbf{3.00 \cdot 10^{-3}}$
Multi_10	$5.00 \cdot 10^{-3}$	18.1	$1.48 \cdot 10^{3}$	1.00	$1.00 \cdot 10^{-3}$	$4.00 \cdot 10^{-3}$
Mono_20	$6.50 \cdot 10^{-2}$	**15.4**	$1.78 \cdot 10^{3}$	$9.28 \cdot 10^{-1}$	$8.00 \cdot 10^{-3}$	$1.70 \cdot 10^{-2}$
Multi_20	$\mathbf{3.70 \cdot 10^{-2}}$	22.3	$1.87 \cdot 10^{3}$	$\mathbf{8.95 \cdot 10^{-2}}$	$\mathbf{4.50 \cdot 10^{-3}}$	$\mathbf{1.30 \cdot 10^{-2}}$

	F7	F8	F9	F10	F11
Mono_5	$4.02 \cdot 10^{-2}$	$1.93 \cdot 10^{11}$	75.3	$1.84 \cdot 10^{-3}$	74.0
Multi_5	$\mathbf{1.37 \cdot 10^{-2}}$	$\mathbf{1.41 \cdot 10^{11}}$	**44.9**	$\mathbf{3.46 \cdot 10^{-4}}$	**44.4**
Mono_10	$\mathbf{2.57 \cdot 10^{-2}}$	$1.83 \cdot 10^{11}$	84.2	$\mathbf{1.32 \cdot 10^{-3}}$	82.4
Multi_10	$2.99 \cdot 10^{-2}$	$\mathbf{1.58 \cdot 10^{11}}$	**63.0**	$4.60 \cdot 10^{-3}$	**61.9**
Mono_20	$1.77 \cdot 10^{-1}$	$2.04 \cdot 10^{11}$	$2.08 \cdot 10^{2}$	$6.24 \cdot 10^{-2}$	$2.08 \cdot 10^{2}$
Multi_20	$\mathbf{3.83 \cdot 10^{-2}}$	$\mathbf{1.73 \cdot 10^{11}}$	**77.8**	$2.76 \cdot 10^{-2}$	**76.3**

Table 5.16: Percentage of saved evaluations by the best multiobjectivisation

	F1	F2	F3	F4	F5	F6
$D = 50$	26.1%	45.9%	40.0%	-28.9%	33.3%	32.6%
$D = 500$	26.9%	-7.7%	25.0%	-21.2%	29.4%	28.0%

	F7	F8	F9	F10	F11
$D = 50$	29.2%	45.5%	31.0%	23.5%	33.3%
$D = 500$	25.9%	38.7%	29.4%	35.3%	29.4%

Mono-objective and multiobjectivised techniques have been compared in terms of the achieved fitness. The median of the error achieved by the best mono-objective and multiobjectivised approaches, for each population size, is given in Table 5.13, considering $D = 50$. The error has been defined as the difference between the achieved fitness and the optimal fitness. It has been calculated considering an accuracy equal to $1 \cdot 10^{-6}$. Moreover, a statistical analysis between the best mono-objective approach and the best multiobjectivised one has been carried out for each population size. For cases in which differences have been statistically significant, data of the best of both algorithms is shown in bold. Table 5.15 shows the same information for the case of $D = 500$. For both values of D, algorithms with lower population sizes have obtained lower errors. In addition, the superiority of the multiobjectivised approaches has been clearly demonstrated. However, in some cases the best mono-objective approach has obtained statistically better results than the best multiobjectivised technique. From a total number of 66 statistical tests, in 41 of them the best multiobjectivised approach has been statistically superior to the best mono-objective approach. The best mono-objective algorithm has been statistically superior in 9 tests. Finally, 16 statistical tests have shown no significant differences between both strategies. In addition, it is important to know which kinds of problems are more adequate to be solved by multiobjectivised approaches. Multiobjectivisation has provided more benefits when it has been applied to a unimodal problem. In fact, for a fixed population size, the best multiobjectivised approach has been statistically better than the best mono-objective algorithm in a 76.2% of the cases in which unimodal problems have been considered. This ratio has decreased to a 37.5% when multimodal problems have been taken into account.

The previous analysis has demonstrated the validity of multiobjectivisation in terms of the achieved quality level. However, it is important to quantify the improvement that can be achieved by using multiobjectivisation, in terms of the invested num-

ber of evaluations. Comparisons have been performed considering the evaluations required to obtain a 50% of success ratio for achieving a quality level. The quality level has been fixed so that all executions have been able to achieve it. The number of evaluations required to achieve a 50% of success ratio have been calculated for the best-behaved mono-objective and multiobjectivised approaches, regardless of the population size. Table 5.16 shows the percentage of saved evaluations by the best multiobjectivised approach in relation to the best mono-objective one. A negative value means that the mono-objective algorithm has converged faster than the corresponding multiobjectivised strategy to the fixed quality level. Once again, the superiority of multiobjectivisation can be noted. The best-behaved multiobjectivised approach has provided benefits in 19 cases from a total number of 22. Moreover, in the majority of the cases, the percentage of saved evaluations has been large. In fact, in some cases, about a 45% of evaluations have been saved.

Second Experiment: On the usage of thresholds

The second experiment has focused on analysing the novel approach based on multiobjectivisation with parameters. Specifically, the considered benchmark problems have been multiobjectivised with the DCN-THR strategy. The aim of the analysis has been to discover the relationship between the values of the threshold ratio th, and the quality of the obtained results. The multiobjectivised versions of the benchmark problems have been solved with the NSGA-II. The parameterisation has been as follows. The number of decision variables D has been fixed to 500. Since lower errors have been obtained with lower population sizes in the previous experiments, the population size has been fixed to 5 individuals. The following values have been used for the threshold ratio th: 0, 0.2, 0.4, 0.6, and 0.8. Finally, the stopping criterion has been fixed to a total number of $2 \cdot 10^7$ evaluations.

Table 5.17 shows, for each benchmark problem, the threshold ratio of the configuration which has obtained the best median of the fitness in $1.25 \cdot 10^6$ evaluations. It also shows the threshold ratios of those configurations which have not been statistically different than the best one. There has been no value which has been able to be among the best ones for every problem. Thus, in order to decide the proper value to use, *a priori* information of the problem to solve is required.

In order to measure the impact over the performance, the RLDs have been used. The quality level has been fixed as the median of the fitness achieved by the best configuration in $1.25 \cdot 10^6$ evaluations. Table 5.18 shows, for each threshold value, the number of evaluations required to obtain a success ratio of 50%, when the aforementioned quality level has been taken into account. Considering the number of evaluations required by suboptimal approaches, the importance of correctly fixing

Table 5.17: Statistical comparison among different threshold values

	F1	F2	F3	F4	F5	F6
Best Threshold	0	0.8	0.2, 0.4	0.4, 0.6	0	0, 0.2

	F7	F8	F9	F10	F11
Best Threshold	0.2	0, 0.2	0.8	0	0.8

Table 5.18: Number of evaluations required to achieve a fixed quality level

	0	0.2	0.4	0.6	0.8
F1	$\mathbf{1.25 \cdot 10^6}$	$2.2 \cdot 10^6$	$3.05 \cdot 10^6$	$4.7 \cdot 10^6$	$6.05 \cdot 10^6$
F2	$1.35 \cdot 10^6$	$1.35 \cdot 10^6$	$1.4 \cdot 10^6$	$1.4 \cdot 10^6$	$\mathbf{1.25 \cdot 10^6}$
F3	$1.75 \cdot 10^6$	$\mathbf{1.25 \cdot 10^6}$	$1.35 \cdot 10^6$	$1.7 \cdot 10^6$	$1.8 \cdot 10^6$
F4	$2.15 \cdot 10^6$	$1.45 \cdot 10^6$	$\mathbf{1.25 \cdot 10^6}$	$\mathbf{1.25 \cdot 10^6}$	$1.5 \cdot 10^6$
F5	$\mathbf{1.25 \cdot 10^6}$	$4 \cdot 10^6$	$5.15 \cdot 10^6$	$6.9 \cdot 10^6$	$1.74 \cdot 10^7$
F6	$1.25 \cdot 10^6$	$\mathbf{1.2 \cdot 10^6}$	$1.95 \cdot 10^6$	$2.5 \cdot 10^6$	$3.75 \cdot 10^6$
F7	$2.3 \cdot 10^6$	$\mathbf{1.25 \cdot 10^6}$	$2 \cdot 10^6$	$3.85 \cdot 10^6$	$8.6 \cdot 10^6$
F8	$\mathbf{1.25 \cdot 10^6}$	$1.3 \cdot 10^6$	$1.45 \cdot 10^6$	$1.4 \cdot 10^6$	$1.45 \cdot 10^6$
F9	$1.9 \cdot 10^6$	$1.85 \cdot 10^6$	$1.75 \cdot 10^6$	$1.65 \cdot 10^6$	$\mathbf{1.25 \cdot 10^6}$
F10	$\mathbf{1.25 \cdot 10^6}$	$1.8 \cdot 10^6$	$2.4 \cdot 10^6$	$3.25 \cdot 10^6$	$5 \cdot 10^6$
F11	$1.85 \cdot 10^6$	$1.85 \cdot 10^6$	$1.85 \cdot 10^6$	$1.6 \cdot 10^6$	$\mathbf{1.25 \cdot 10^6}$

the parameter is clear. For instance, considering the problem F5, the number of evaluations required by the best configuration approximately represents a 7.18% of the evaluations required by the worst one.

Another open research question is whether the proper value of th depends on the optimisation stage or not. With the aim of answering this question the following experiment has been performed. First, four different optimisation stages have been defined. To define such stages the median of the fitness values achieved by the best configuration in $2.5 \cdot 10^5$, $5 \cdot 10^5$, $7.5 \cdot 10^5$, and $1 \cdot 10^6$ evaluations have been considered. Then, for each calculated value, 30 individuals with a quality similar to such a value have been generated. In order to generate the 30 individuals, the best configuration has been executed 30 times. For each one of these 30 executions, the generated individual has been the first one that has achieved an original objective value lower than the fixed value for the corresponding stage.

Table 5.19: Statistical comparison among different threshold values by stages - F4

	Stage 0					Stage 1				
	0	0.2	0.4	0.6	0.8	0	0.2	0.4	0.6	0.8
0	↔	↔	↔	↔	↓	↔	↔	↔	↓	↓
0.2	↔	↔	↔	↔	↓	↔	↔	↔	↓	↓
0.4	↔	↔	↔	↔	↓	↔	↔	↔	↔	↔
0.6	↔	↔	↔	↔	↔	↑	↑	↔	↔	↔
0.8	↑	↑	↑	↔	↔	↑	↑	↔	↔	↔
	Stage 2					Stage 3				
	0	0.2	0.4	0.6	0.8	0	0.2	0.4	0.6	0.8
0	↔	↔	↔	↔	↔	↔	↑	↑	↑	↑
0.2	↔	↔	↔	↔	↔	↓	↔	↔	↔	↔
0.4	↔	↔	↔	↔	↔	↓	↔	↔	↔	↔
0.6	↔	↔	↔	↔	↔	↓	↔	↔	↔	↔
0.8	↔	↔	↔	↔	↔	↓	↔	↔	↔	↔

Table 5.20: Statistical comparison among different threshold values by stages - F5

	Stage 0					Stage 1				
	0	0.2	0.4	0.6	0.8	0	0.2	0.4	0.6	0.8
0	↔	↔	↔	↔	↔	↔	↔	↑	↑	↑
0.2	↔	↔	↔	↑	↑	↔	↔	↔	↑	↑
0.4	↔	↔	↔	↔	↑	↓	↔	↔	↔	↔
0.6	↔	↓	↔	↔	↔	↓	↓	↔	↔	↔
0.8	↔	↓	↓	↔	↔	↓	↓	↔	↔	↔
	Stage 2					Stage 3				
	0	0.2	0.4	0.6	0.8	0	0.2	0.4	0.6	0.8
0	↔	↑	↑	↑	↑	↔	↑	↑	↑	↑
0.2	↓	↔	↔	↔	↔	↓	↔	↔	↔	↔
0.4	↓	↔	↔	↔	↔	↓	↔	↔	↔	↔
0.6	↓	↔	↔	↔	↔	↓	↔	↔	↔	↔
0.8	↓	↔	↔	↔	↔	↓	↔	↔	↔	↔

Afterwards, the NSGA-II has been tested with different values of the threshold ratio for each considered stage. Considering each one of the stages, each one of the executions of the NSGA-II has included one of the 30 individuals generated by the

Table 5.21: Statistical comparison among different threshold values by stages - F11

	Stage 0					Stage 1				
	0	0.2	0.4	0.6	0.8	0	0.2	0.4	0.6	0.8
0	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔
0.2	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔
0.4	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔
0.6	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔
0.8	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔
	Stage 2					Stage 3				
	0	0.2	0.4	0.6	0.8	0	0.2	0.4	0.6	0.8
0	↔	↔	↔	↔	↔	↔	↔	↔	↔	↓
0.2	↔	↔	↔	↔	↓	↔	↔	↔	↔	↓
0.4	↔	↔	↔	↔	↔	↔	↔	↔	↔	↓
0.6	↔	↔	↔	↔	↔	↔	↔	↔	↔	↓
0.8	↔	↑	↔	↔	↔	↑	↑	↑	↑	↔

aforementioned method in its initial population. The remaining individuals of the population have been randomly generated. By this way, the performance of the NSGA-II with different threshold values can be tested when it starts from different quality levels. The stopping criterion has been fixed to $5 \cdot 10^5$ evaluations. The different configurations of the NSGA-II have been compared in terms of the achieved fitness. Tables 5.19, 5.20, and 5.21 show the comparison among the considered threshold values for the problems F4, F5, and F11, respectively. For each stage, they show if the row configuration is statistically better (↑), not different (↔), or worse (↓), than the corresponding column configuration. For each problem, the statistical tests show that differences among configurations depend on the considered optimisation stage. For instance, taking into account the stage number 0 of the problem F4, the configuration that uses $th = 0.8$ is better than the one that uses $th = 0$. However, in the stage number 3 the configuration with $th = 0$ performs better than the configuration with $th = 0.8$. In the case of the problem F5, $th = 0$ seems to be the most appropriate value for the whole run because there is not a better value in any of the analysed stages. For the same reasons, in the problem F11 the value $th = 0.8$ seems to be the most appropriate value.

Third Experiment: Adaptive multiobjectivisation

The last experiment has been devoted to analyse the adaptive multiobjectivisation. The hyperheuristics were executed considering as low-level configurations the schemes analysed in the previous experiment. Specifically, 11 low-level configurations were defined. The difference among them was the definition of the threshold value of DCN-THR. The threshold values were evenly distributed in the range $[0, 1]$.

Initial experiments showed that the low values of k were the most adequate ones, and that differences between MONO_WEIGHT and ELI-MONO_WEIGHT were not large. However, the ELI-MONO_WEIGHT obtained slightly better results in most of the cases. In order to show the benefits of the adaptive multiobjectivisation, ELI-MONO_WEIGHT with $k = 5$ was executed with the F problems. The stopping criterion was fixed to 2.500.000 function evaluations. The low-level configurations were also independently executed. Figures 5.14 and 5.15 show the median of the fitness obtained by the low-level configurations with the different values of th. The median of the fitness obtained with ELI-MONO_WEIGHT is also shown. It can be observed that in every problem the hyperheuristic has been able to detect which are the best threshold values. The results attained by the hyperheuristic are very similar to the ones obtained by the best low-level configuration. In fact, in two of the problems (F4 and F5) the median obtained by ELI-MONO_WEIGHT has been better than the one obtained by the best low-level configuration. In such cases, differences with the best sequential approach were not statistically significant. Anyway, the advantages of using the hyperheuristic are clear.

It is also important to measure the amount of resources that can be saved (or lost) with the adaptive multiobjectivisation. In order to give a measure of the impact produced by the hyperheuristics, the factors between the evaluations required to obtain a 50% of success ratio with the models that use fixed thresholds and with the ELI_MONO-WEIGHT model have been calculated. The desired fitness has been established as the highest median obtained by the involved models. Tables 5.22 and 5.23 show such values. In some cases large values have appeared. This means that by using the hyperheuristic a large amount of resources can be saved, when compared with fixed inappropriate threshold values.

Finally, it is worthy to note that the schemes that used $th = 0$ obtained, in general, a competitive median value. In six of the problems the value $th = 0$ was the best one, and in the rest of the problems competitive results were attained. Therefore, the advantage of using the ELI-MONO_WEIGHT when compared to the configuration with $th = 0$ is not so clear. However, additional tests have shown that in many cases, the value $th = 0$ is not adequate. Specifically, the ELI-MONO_WEIGHT was executed with the same low-level configurations than in the previous experiment, but

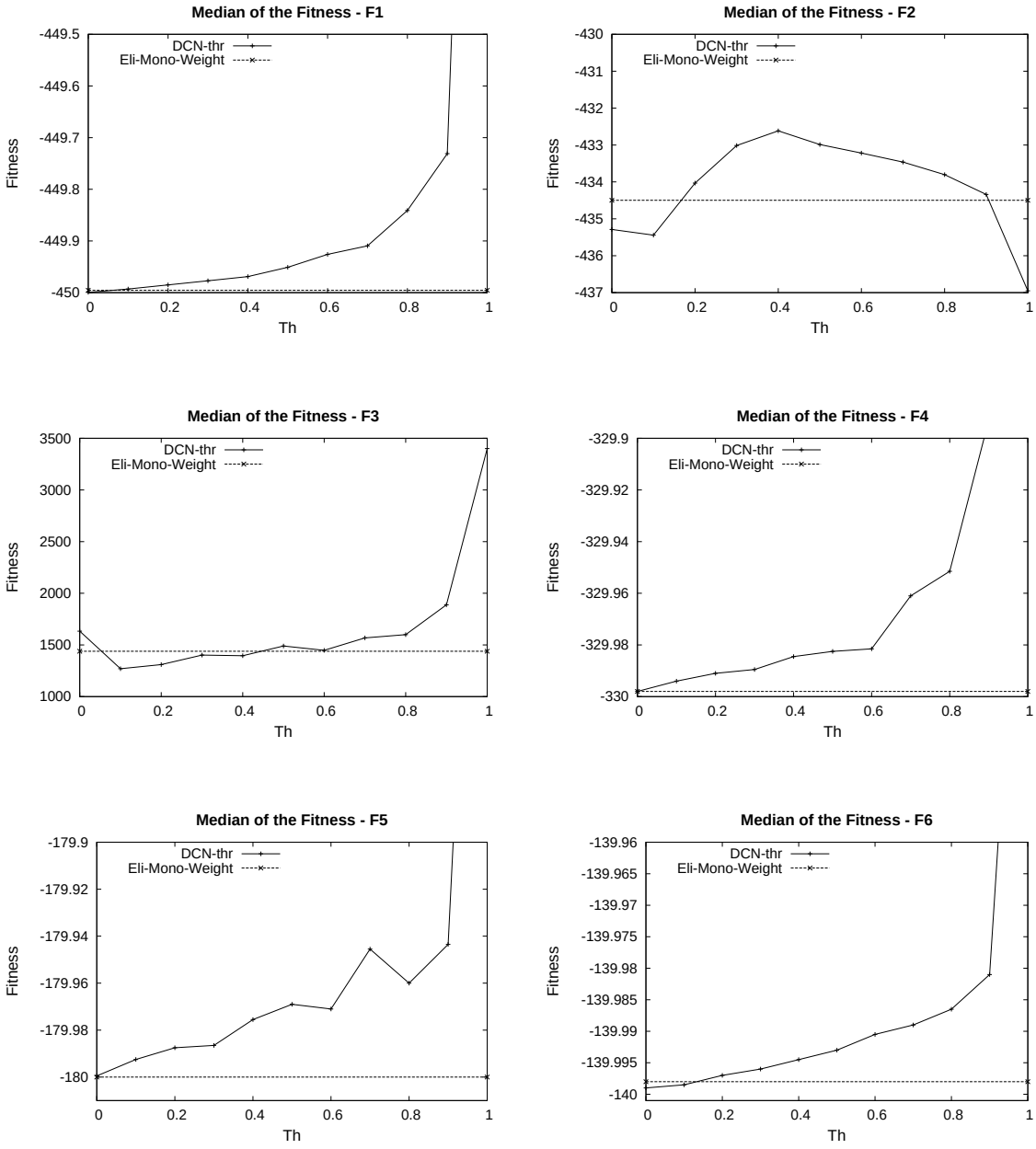


Figure 5.14: Median of the fitness considering different multiobjectivisations (F1 - F6)

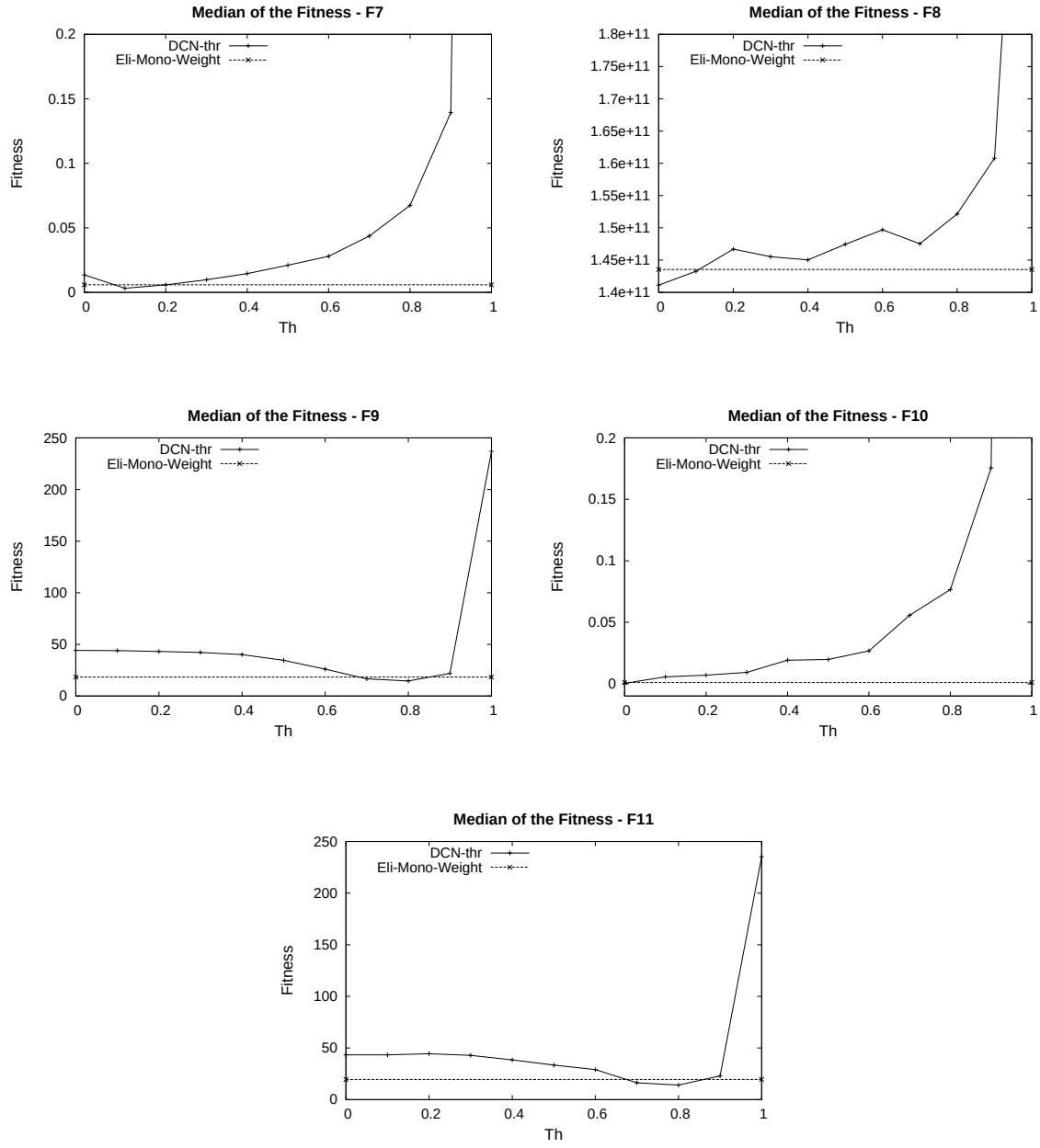


Figure 5.15: Median of the fitness considering different multiobjectivisations (F7 - F11)

Table 5.22: Factor of additional evaluations performed with each th in comparison to ELI-MONO-WEIGHT (F1 - F6)

	F1	F2	F3	F4	F5	F6
$th = 0$	0.70	0.96	1.25	1.08	0.75	0.85
$th = 0.1$	1.03	0.96	0.76	1.20	1.56	0.78
$th = 0.2$	1.23	1.03	0.82	1.28	1.66	1.11
$th = 0.3$	1.28	1.10	0.95	1.33	1.81	1.29
$th = 0.4$	1.47	1.13	0.95	1.44	2.14	1.47
$th = 0.5$	1.60	1.10	1.06	1.44	2.17	1.60
$th = 0.6$	1.74	1.08	1.01	1.48	2.37	1.67
$th = 0.7$	1.83	1.07	1.15	1.61	2.79	1.89
$th = 0.8$	2.11	1.03	1.20	1.71	2.60	2.13
$th = 0.9$	2.33	1.01	1.53	2.04	3.01	2.50
$th = 1$	4.71	0.80	3.12	2.80	6.25	5.20

Table 5.23: Factor of additional evaluations performed with each th in comparison to ELI-MONO-WEIGHT (F7 - F11)

	F7	F8	F9	F10	F11
$th = 0$	1.47	1.00	1.45	0.84	1.41
$th = 0.1$	0.70	1.02	1.45	1.31	1.41
$th = 0.2$	1.00	1.05	1.43	1.36	1.42
$th = 0.3$	1.29	1.05	1.42	1.44	1.40
$th = 0.4$	1.54	1.04	1.38	1.63	1.34
$th = 0.5$	1.76	1.06	1.30	1.71	1.25
$th = 0.6$	1.91	1.10	1.16	1.75	1.18
$th = 0.7$	2.27	1.06	0.96	2.06	0.94
$th = 0.8$	2.60	1.13	0.86	2.25	0.84
$th = 0.9$	3.37	1.24	1.07	2.95	1.05
$th = 1$	8.92	3.33	4.03	5.10	3.96

Table 5.24: Median of the fitness with a population with 100 individuals (F1 - F6)

	F1	F2	F3	F4	F5	F6
$th = 0$	-445.91	-418.09	4089.29	-314.62	-179.58	-139.76
ELI-MONO-WEIGHT	-449.62	-421.70	3460.63	-329.62	-179.96	-139.96

Table 5.25: Median of the fitness with a population with 100 individuals (F1 - F6)

	F7	F8	F9	F10	F11
$th = 0$	0.61	2.096e11	325.27	3.02	333.62
ELI-MONO-WEIGHT	0.11	2.025e11	295.30	0.13	292.28

considering a population with 100 individuals. The NSGA-II with DCN-THR, $th = 0$ and $N = 100$ was also executed. Tables 5.24 and 5.25 show the median obtained by ELI-MONO_WEIGHT and by the model with $th = 0$ in 2.500.000 function evaluations. In every problem the median obtained by the adaptive scheme is better. Moreover, the statistical tests confirm the superiority of ELI-MONO_WEIGHT for every problem, except for F8.

5.2 Multi-objective Benchmark Problems

5.2.1 Problems Description

Testing multi-objective algorithms in a systematic way is very important, so there have been several attempts to define test suites or toolkits for building test suites. An extensive review of a large amount of multi-objective test problems was presented in [130]. The authors detected several drawbacks that affected some of the most-known benchmarks problems. Considering it, they proposed a new toolkit for creating multi-objective optimisation problems that has become very popular. Among the problems proposed in the literature, some of the most popular ones are the following:

- ZDT problems [255]: It is a well known suite of six test problems created by Zitzler *et al.* The test problems consist of two optimisation objectives. Five of the problems are real-valued problems, while ZDT5 is a binary encoded one. One of the main drawbacks is that the first objective function only depends on one parameter. In addition, none of the problems are non-separable.

- DTLZ problems [79]: It is a suite of benchmark problems created by Deb *et al.* The defined problems are scalable to any number of objectives. The suite consists of seven non-constrained problems, and two problems with side constraints. One of the main drawbacks is that none of the problems are deceptive, and none of the problems are non-separable.

Table 5.26: Main features of WFG problems

Problem	Obj.	Separability	Modality	Bias	Geometry
WFG1	$f_{1:M}$	S	U	polynomial, flat	convex, mixed
WFG2	$f_{1:M-1}$	NS	U	-	convex, disconnected
	f_M	NS	M		
WFG3	$f_{1:M}$	NS	U	-	linear, degenerate
WFG4	$f_{1:M}$	S	M	-	concave
WFG5	$f_{1:M}$	S	D	-	concave
WFG6	$f_{1:M}$	NS	U	-	concave
WFG7	$f_{1:M}$	S	U	parameter dep.	concave
WFG8	$f_{1:M}$	NS	U	parameter dep.	concave
WFG9	$f_{1:M}$	NS	D	parameter dep.	concave

- Van Veldhuizen's Test Suite [239]: It is a group of problems defined by different authors that were extensively used prior to the publications of the ZDT and DTLZ problems. Among the problems, there are seven without constraints. The main drawbacks of the problems are that they are not scalable, and that they have few parameters. In addition, none of the problems are deceptive.

- WFG toolkit [130]: It is a toolkit proposed by Huband *et al.* for constructing well designed multi-objective test problems. Such a toolkit was used to create nine scalable test problems, known as WFG1 - WFG9, which are nowadays very popular.

Huband *et al.* analysed the different features of the optimisation problems that may affect the behaviour of multi-objective solvers. Most of the analysed features are similar than the ones studied in mono-objective optimisation: modality, deception and separability. In addition, other features were identified:

- Pareto front geometry: Usually, practitioners distinguish between convex and concave Pareto geometries. In addition, the continuity of the Pareto front is usually analysed. Finally, the Pareto front is said to be degenerate if its dimension is lower than the objective space in which it is embedded less one.

- Bias: A multi-objective problem has bias if the density variation in the space of the objectives is low, when an even spread of solutions in the space of the variables is considered.

Table 5.26 summarises the main features of WFG problems. The properties regarding separability (separable or non-separable), modality (unimodal, multimodal or deceptive), bias, and geometry of the Pareto Front are presented. It can be observed that they cover several different characteristics. For this reason, the WFG problems have been selected to test the performance of the defined hyperheuristic. Both the sequential and parallel version has been analysed.

5.2.2 Experimental Evaluation

In this section the experimental evaluation performed with the multi-objective hyperheuristic proposed in this research is presented. The optimisation schemes have been implemented using METCO. Considering the description exposed in the previous section, the WFG1-WFG9 multi-objective benchmark problems have been used to validate the proposal.
The main aim of the experimental evaluation has been to demonstrate that the defined multi-objective hyperheuristic can automatically distribute the computational resources in an intelligent way among a set of low-level heuristics, and to demonstrate that in some cases, the hyperheuristic might even outperform the best low-level considered approaches. In addition, several experiments devoted to analyse the behaviour of the parallel version of the hyperheuristic have been performed. In this case, an analysis of the influence that the migration stage has over the performance of the parallel approach has been carried out. In addition, the scalability of the parallel proposal has been analysed. Since experiments have involved the use of stochastic algorithms, each execution has been repeated 30 times.

First Experiment: Performance vs. Amount of Low-level Approaches

In order to execute the multi-objective hyperheuristic, a set of low-level approaches (or configurations) must be defined. The performance of the proposal might depend on the amount of low-level approaches that are considered. For this reason, the first experiment has been devoted to analyse the performance of the hyperheuristic when

different quantities of low-level configurations have been considered. Experiments with up to 128 low-level configurations have been carried out. The low-level configurations have been based on using the NSGA-II approach with a fixed variation stage. Specifically, the UM and the SBX operators have been used. A direct encoding based on real-valued genes has been considered. In every case, the probability of crossover has been fixed to 0.9, while the population size has been fixed to 100. Finally, in order to define several low-level configurations, several values of p_m have been considered.

In order to better inspect the behaviour of the hyperheuristic, the results obtained by the low-level approaches must be analysed. For this reason, the low-level approaches were firstly independently executed. In such executions, the stopping criterion was fixed to a total number of 1.000.000 function evaluations. Figure 5.16 and 5.17 show the median of the hypervolume obtained with different values of p_m for each of the considered benchmark problems. In every case the low values of p_m are more adequate. In fact, if the lowest values of p_m are discarded (the ones lower than $\frac{1}{D}$, being D the number of variables) the functions are practically monotonically decreasing.

In order to facilitate the analysis of the hyperheuristic, the probabilities considered for the low-level configurations have been in the range $[\frac{1}{D}, 1]$. Specifically, in order to define a set of C low-level configurations, C values evenly distributed in the specified range have been considered. In this way, it is very easy to check whether the hyperheuristic is assigning the resources correctly or not. It is important to remark that in some cases, the best values of p_m might depend on the stage of the optimisation process. In order to avoid inconsistencies in the analysis, comparisons have taken into account both the quality obtained by the different models and the way in which the resources have been granted.

Two adaptive schemes have been considered. The first one assigns the resources using the multi-objective hyperheuristic HV_WEIGHT. The local stopping criterion has been fixed to 5.000 evaluations, i.e. each 5.000 evaluations the hyperheuristic selects which low-level configuration must keep executing. In order to decide the values of the parameters of the hyperheuristic some preliminary executions were performed. The value of β has been fixed in a way that the 10% of the decisions performed by the hyperheuristic follows a uniform distribution, i.e., $\beta * n_h = 0.1$. The value of k has been fixed to ∞. The second scheme (RANDOM) has been based on assigning the resources randomly. As in the first scheme, the decisions are taken each 5.000 evaluations. In both cases, the global stopping criterion has been fixed to a total number of 1.000.000 function evaluations. Figures 5.18 and 5.19 show, for each benchmark problem, the median of the hypervolume achieved at the end of the executions. Results are shown both for the HV_WEIGHT and RANDOM selection

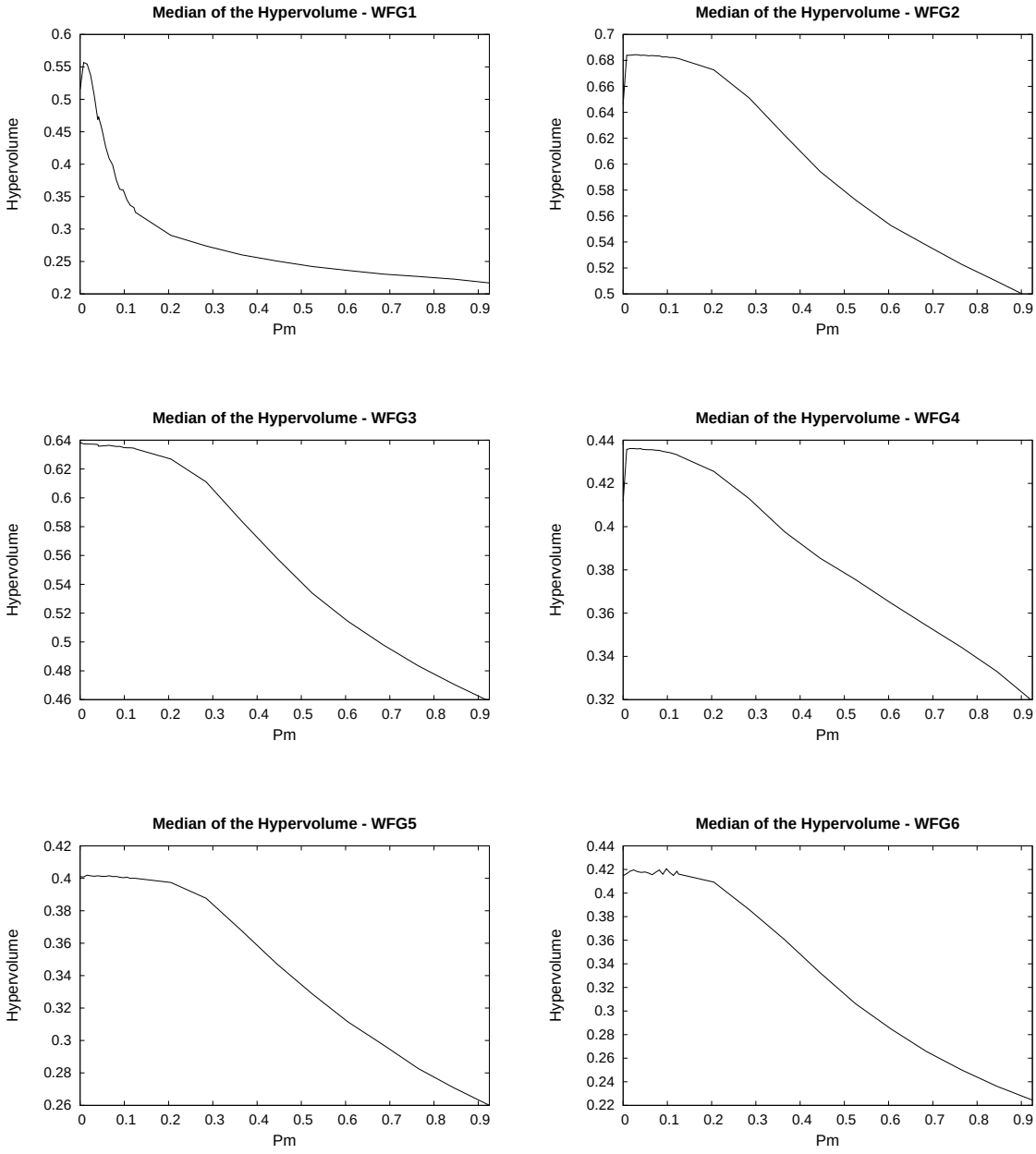


Figure 5.16: Median of the hypervolume considering different mutation probabilities (WFG1 - WFG6)

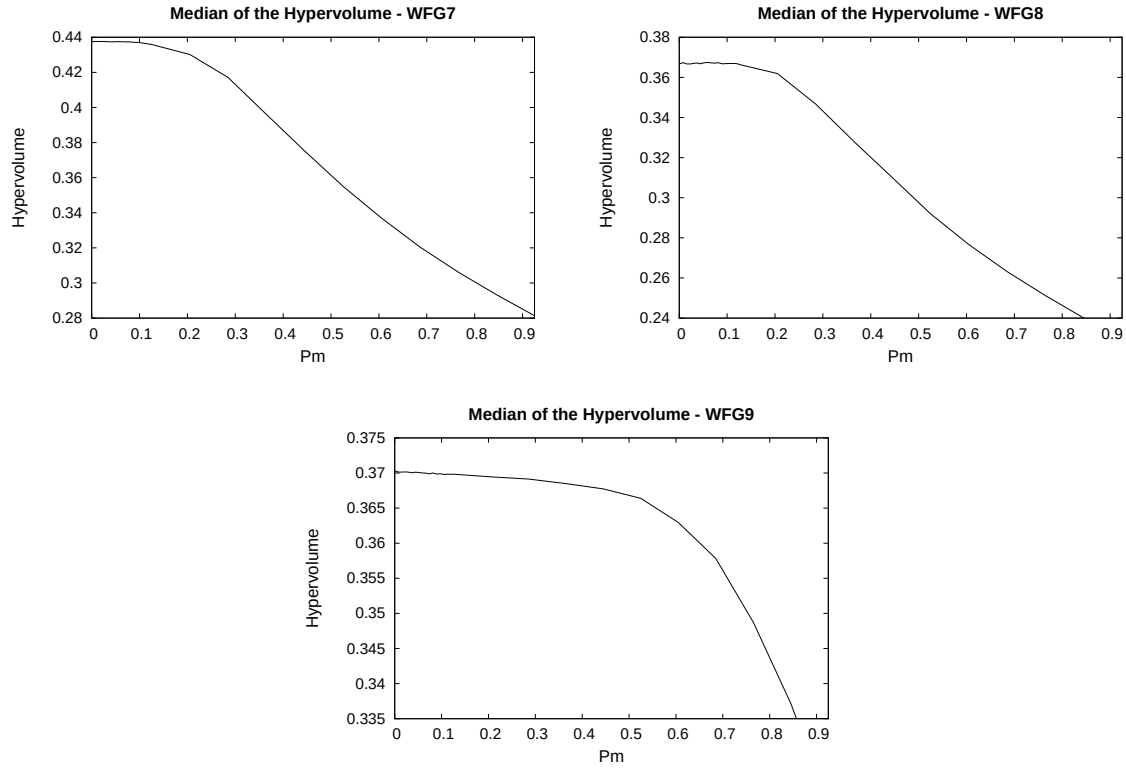


Figure 5.17: Median of the hypervolume considering different mutation probabilities (WFG7 - WFG9)

methods. Tests with up to 128 low-level configurations have been carried out. It can be noted that in most of the problems the advantages of using HV_WEIGHT are clear. Thus, assigning the resources using the designed methodology is preferable to assigning them in a random way. In the case of the WFG9 problem the results are not conclusive. WFG9 is highly deceptive and contains the hardest non-separable components [130]. For this reason, the low-level approaches are not able to escape - in most cases - from the local optima. Thus, independently of the way in which the resources were shared among the low-level approaches, very low quality values were obtained in some of the runs. In addition, it is important to remark that in most of the cases the differences between the qualities achieved by the considered models with 128 configurations are not as large as when fewer configurations have been used. The reason is that when 128 configurations have been used, a total number of 640.000 evaluations have been utilised to execute each low-level configuration one

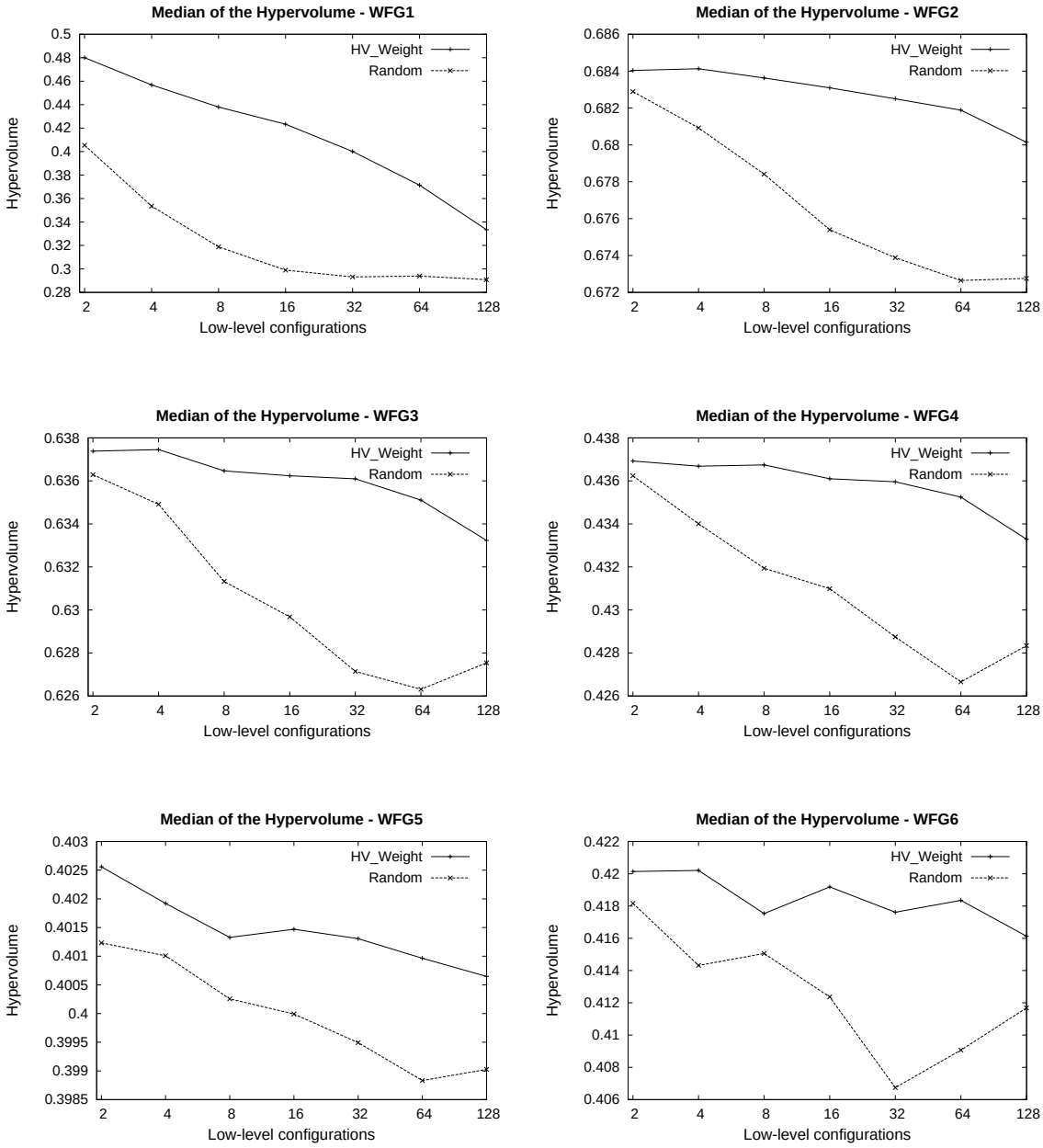


Figure 5.18: Median of the hypervolume considering different number of configurations (WFG1 - WFG6)

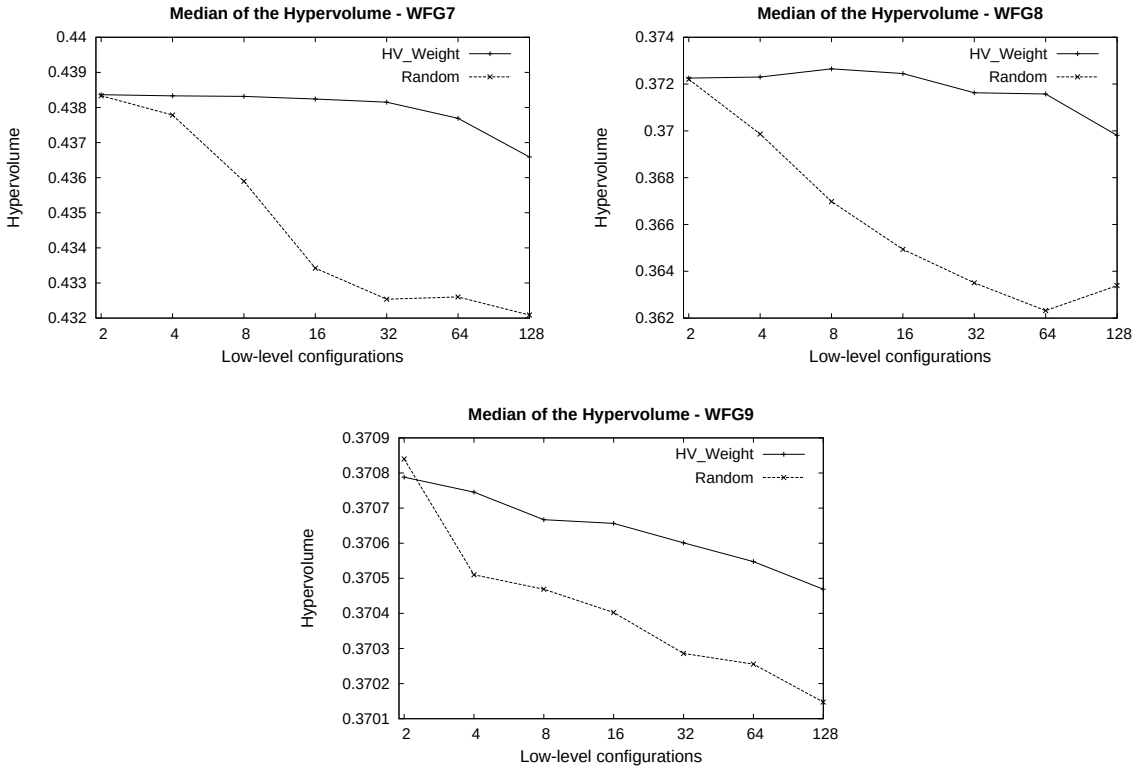


Figure 5.19: Median of the hypervolume considering different number of configurations (WFG7 - WFG9)

time. Thus, a lot of evaluations are used in the initial stage where every low-level approach is executed. Anyway, even in such a case, the HV_WEIGHT model has obtained higher hypervolume values than the RANDOM approach.

It can also be observed that as more configurations are added, the hypervolume tends to decrease. As more configurations are added, the initial stage in which each configuration is tested is larger. Thus, more resources are granted to non-optimal configurations. In addition, due to the way in which the low-level configurations have been defined, in every scheme there is at least one adequate low-level approach. Therefore, the obtained quality decreases as more configurations are added. It is important to remark that this might not happen in every case. For instance, a model with few configurations in which none of them are adequate, will behave worse than a model with many configurations, in which some of them are suitable for the problem. For this reason, the addition of alternative configurations might

Table 5.27: Statistical comparison between HV_WEIGHT and RANDOM (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
2 *conf.*	↑	↑	↑	↑	↑
4 *conf.*	↑	↑	↑	↑	↑
8 *conf.*	↑	↑	↑	↑	↑
16 *conf.*	↑	↑	↑	↑	↑
32 *conf.*	↑	↑	↑	↑	↑
64 *conf.*	↑	↑	↑	↑	↑
128 *conf.*	↑	↑	↑	↑	↑

Table 5.28: Statistical comparison between HV_WEIGHT and RANDOM (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
2 *conf.*	↑	↔	↔	↔
4 *conf.*	↑	↑	↑	↔
8 *conf.*	↑	↑	↑	↔
16 *conf.*	↑	↑	↑	↑
32 *conf.*	↑	↑	↑	↔
64 *conf.*	↑	↑	↑	↔
128 *conf.*	↑	↑	↑	↑

increase or decrease the quality of the obtained results.

Tables 5.27 and 5.28 show the results of the statistical comparisons between the models HV_WEIGHT and RANDOM at the end of the executions. For each problem and amount of low-level configurations, a ↑ is shown if the model HV_WEIGHT obtains higher hypervolume values than RANDOM with statistically significant differences. In cases where the differences are not significant the symbol ↔ is used. No cases were found in which the RANDOM method was significantly better than HV_WEIGHT. In most cases, the HV_WEIGHT model is superior to RANDOM. WFG7-9 were the only problems in which HV_WEIGHT was not significantly superior to RANDOM in every considered execution. In the case of WFG7 and WFG8 both approaches reached similar solutions at the end of the executions when two low-level configurations were considered. In fact, in both cases the obtained solutions were good approximations of the Pareto Front, as it can be appreciated in their 50%-attaintment

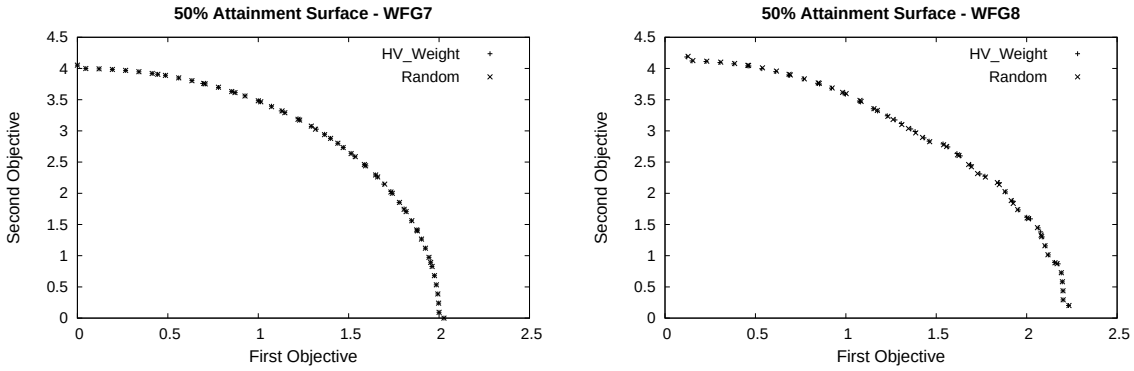


Figure 5.20: 50% Attainment Surfaces at 1.000.000 Evaluations (WFG7, WFG8)

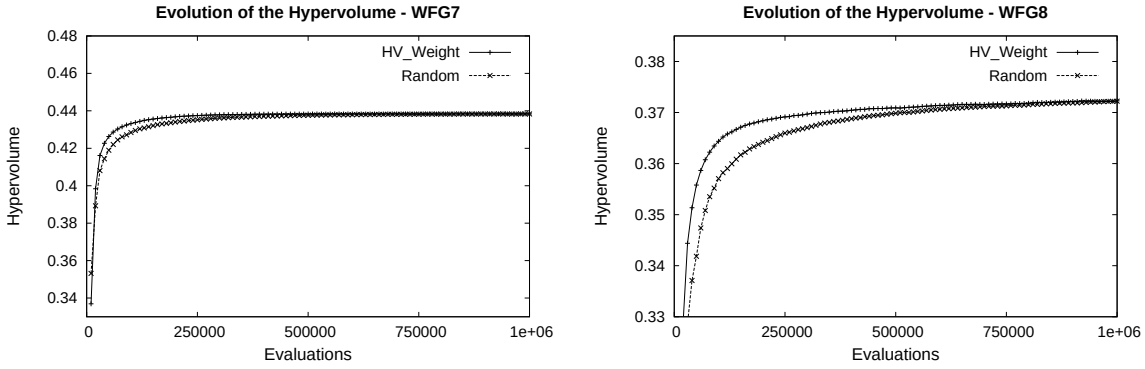


Figure 5.21: Evolution of the median of the hypervolume (WFG7, WFG8)

surfaces (Figure 5.20). The evolution of the hypervolume for both cases is shown in Figure 5.21. Although both approaches reached similar hypervolume values at the end of the executions, the convergence to high hypervolume values is faster when HV_WEIGHT is used. Since there has been no case in which RANDOM has been superior to HV_WEIGHT, the advantages of using HV_WEIGHT are clear.

The previous analysis has shown some of the advantages of HV_WEIGHT. However, it is also important to quantify the improvement that has been achieved. Tables 5.29 and 5.30 show the percentage of evaluations that can be saved by using the HV_WEIGHT model instead of the RANDOM approach. It has been calculated using the evaluations required by each model to achieve a 50% of success ratio. The desired quality hypervolume has been calculated in the following way. First, the

Table 5.29: Percentage of saved evaluations with HV-WEIGHT (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
2 *conf.*	54.54%	36.35%	46.51%	46.39 %	45.00%
4 *conf.*	67.67%	63.00%	58.76%	72.00 %	59.37%
8 *conf.*	73.74%	66.67%	71.72%	74.48 %	77.01%
16 *conf.*	75.51%	68.69%	71.00%	72.00 %	68.60%
32 *conf.*	70.00%	66.67%	71.00%	69.70 %	69.47%
64 *conf.*	55.00%	57.00%	57.00%	57.44 %	56.99%
128 *conf.*	30.00%	27.83%	25.51%	22.82 %	25.00%

Table 5.30: Percentage of saved evaluations with HV-WEIGHT (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
2 *conf.*	72.22%	46.34 %	8.51%	-6.38%
4 *conf.*	86.45%	66.67 %	64.28%	45.83%
8 *conf.*	63.95%	66.67 %	76.59%	66.66%
16 *conf.*	75.55%	72.16 %	74.48%	56.86%
32 *conf.*	75.00%	70.10 %	68.36%	50.98%
64 *conf.*	58.00%	54.54 %	58.58%	27.08%
128 *conf.*	17.39%	26.53 %	29.29%	20.51%

median of the hypervolume obtained by using the HV_WEIGHT and RANDOM models at the end of the executions have been calculated. The lowest value has been considered as the desired quality level. This ensures that both models attain the desired quality level. The saved percentages show that by using the HV_WEIGHT approach, a large amount of resources can be saved. In fact, in some cases, more than 80% of resources can be saved. When many configurations have been used, the saved evaluations have not been so large. The reasons are that in such cases the initial stage where each configuration is executed once is larger. In the case of WFG9 it has appeared a case where the RANDOM model converges faster than HV_WEIGHT. The reasons have been previously explained.

The HV_WEIGHT approach has obtained better results than the RANDOM scheme in terms of the hypervolume of the obtained solutions. This indicates that more resources are given to the most suitable low-level approaches in every case. For the case that considers 16 configurations, the sharing of resources has been measured. Figures 5.22 and 5.23 show the percentage of resources granted to each one of the

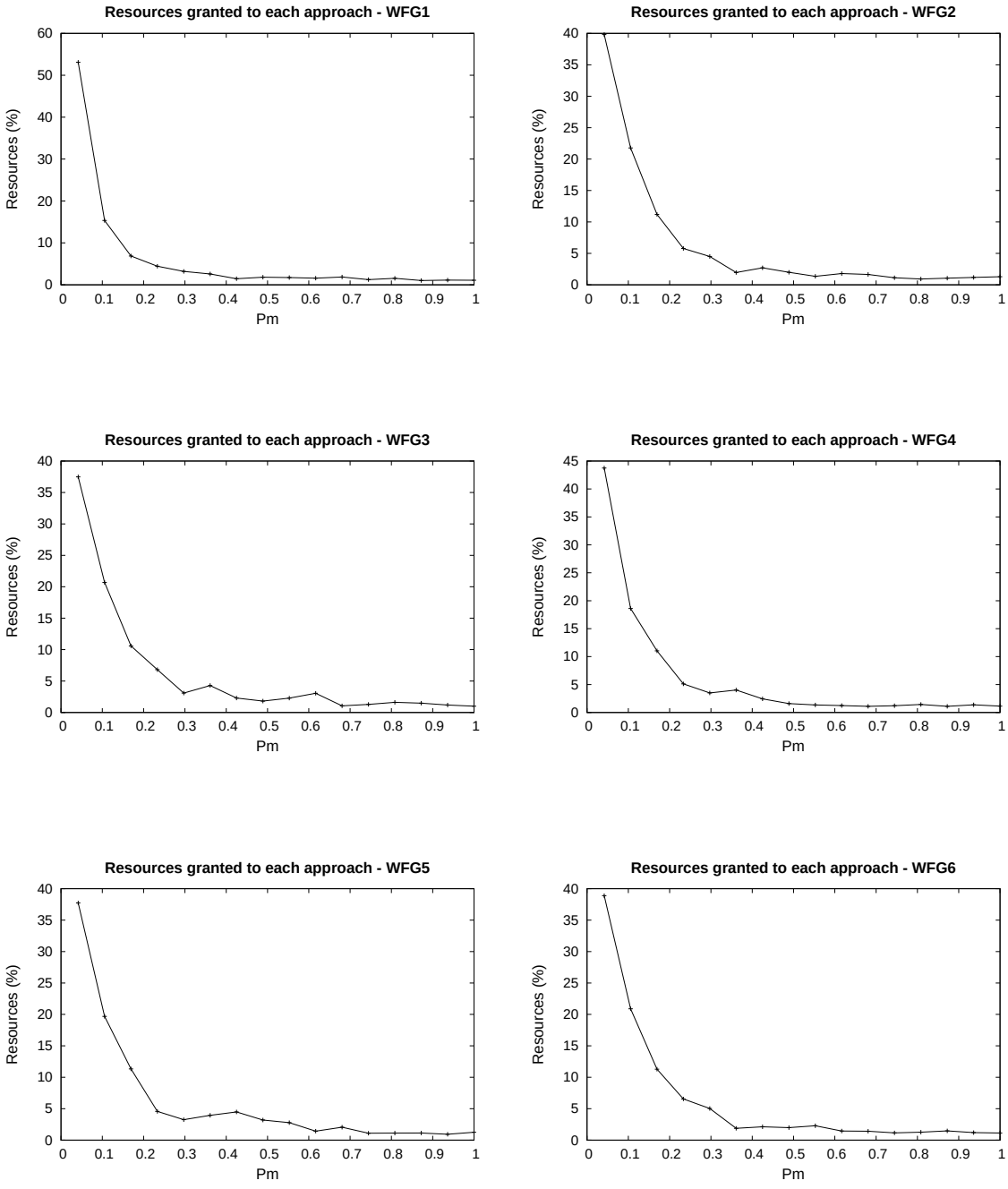


Figure 5.22: Resource sharing with 16 configurations (WFG1 - WFG6)

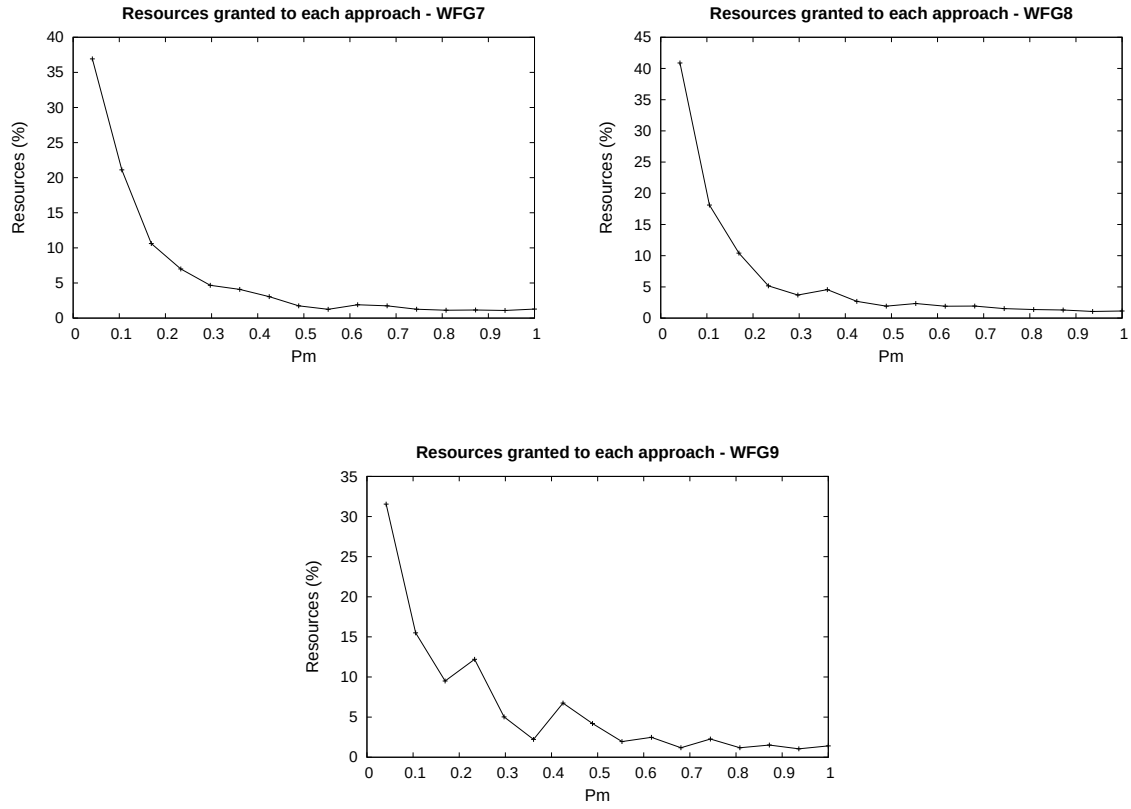


Figure 5.23: Resource sharing with 16 configurations (WFG7 - WFG9)

low-level configurations. It can be appreciated that a higher amount of resources are granted to the low-disruptive configurations in every case. This means that the hyperheuristic is detecting which are the most suitable configurations, and more resources are granted to them.

The increase on the generality of the hyperheuristics is usually achieved at the expense of reduced solution quality when compared to the best low-level approaches. For this reason, it is also important to quantify the relative performance with respect to the best low-level configuration. In order to give a measure of the impact produced by the hyperheuristics, the factors between the evaluations required to obtain a 50% of success ratio with the HV_WEIGHT model and with the best sequential low-level approach have been calculated. The desired quality level was fixed as in the previous experiment. Tables 5.31 and 5.32 show such a value for each problem and for the different number of considered configurations. It can be observed that, as

Table 5.31: Factor of additional evaluations performed with HV-WEIGHT (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
2 *conf.*	0.89	0.80	0.68	0.76	0.79
4 *conf.*	1.20	0.84	0.78	0.91	0.94
8 *conf.*	1.34	0.98	1.36	1.10	0.97
16 *conf.*	1.72	1.38	1.34	1.42	1.02
32 *conf.*	2.06	1.61	1.47	1.50	1.23
64 *conf.*	3.12	1.98	2.50	2.27	1.15
128 *conf.*	5.50	3.44	3.66	4.26	1.79

Table 5.32: Factor of additional evaluations performed with HV-WEIGHT (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
2 *conf.*	0.79	0.64	0.25	0.30
4 *conf.*	1.13	0.74	0.33	0.60
8 *conf.*	2.08	1.17	0.38	0.80
16 *conf.*	1.58	1.43	0.52	1.50
32 *conf.*	2.58	1.79	0.71	2.00
64 *conf.*	2.17	2.35	0.87	2.5
128 *conf.*	3.75	4.16	1.31	3.2

it was expected, in several problems the hyperheuristic requires a larger amount of evaluations (ratios greater than 1) than the corresponding best low-level approach. Moreover, the factors increase as more configurations are considered. However, in every case the factor is much lower than the number of low-level configurations. Since the testing of each low-level configuration can be avoided by using the hyperheuristic, the overall computing and user effort saved is very large.

Second Experiment: Improving on the results obtained with the best low-level approach

In the theoretical analysis, it was shown that in some cases, the hyperheuristics might obtain better values than any of the involved low-level approaches. The previous experimental analysis shows that HV_WEIGHT required fewer evaluations than the best sequential approach in several cases, demonstrating that, in practice, it

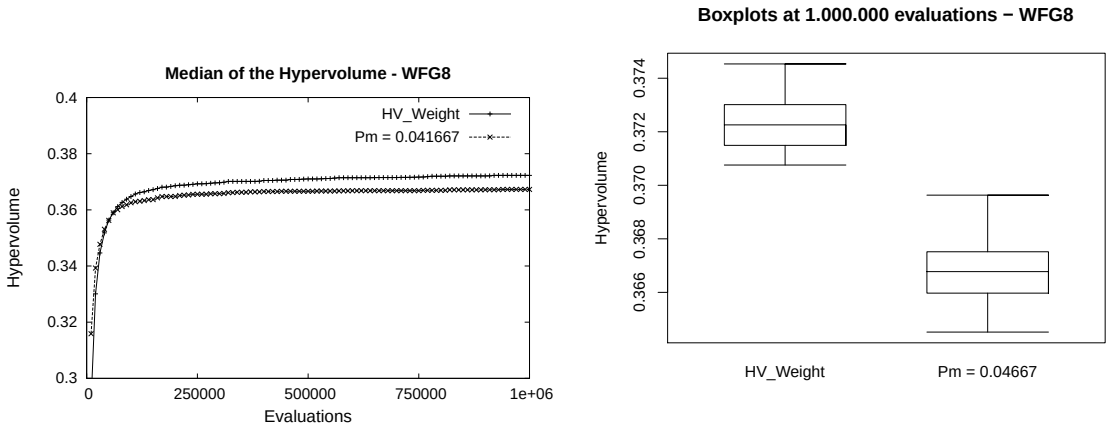


Figure 5.24: Hypervolume for WFG8 with different configurations

also happens. The most outstanding case occurs with WFG8. In such a case the hyperheuristic has been better than the best low-level approach, except in the case of 128 configurations. Moreover, in the case with two low-level configurations the number of evaluations required by the hyperheuristic has been much lower than the evaluations required by the best low-level approach. Figure 5.24 shows the evolution of the median of the hypervolume for the best low-level approach, and for HV_WEIGHT with two low-level configurations. The boxplots at the end of the executions are also shown. It can be appreciated that the hyperheuristic attains much better values. Therefore, the synergy of combining a low-disruptive configuration with a high-disruptive configuration is clear. In addition, the statistical tests confirm the superiority of the hyperheuristic. In fact, the boxplots show that at the end of the executions, every run of the HV_WEIGHT obtained fronts with higher hypervolume values than the ones obtained with the best low-level approach.

Third Experiment: Adaptation level

Previous analysis has shown the advantages of using HV_WEIGHT with a global adaptation level, i.e. with the parameterisation $k = \infty$. It is interesting to analyse the robustness of the approach with respect to the value of k. In order to check it, the model HV_WEIGHT was executed with the WFG tests, but considering the parameterisations $k = \{1, 5, 10, 100, 1000, \infty\}$. In every case, 32 low-level configurations were considered. They were made up as in the previous experiments. The remaining parameters were also the same than in the previous experiments. The

Table 5.33: Median of the hypervolume for different values of k (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
$k = 1$	0.3815	0.6816	0.6316	0.4323	0.4002
$k = 5$	0.4123	0.6836	0.6364	0.4361	0.4009
$k = 10$	0.3900	0.6834	0.6363	0.4359	0.4013
$k = 100$	0.3964	0.6826	0.6359	0.4356	0.4011
$k = 1000$	0.3918	0.6820	0.6358	0.4353	0.4012
$k = \infty$	0.4001	0.6824	0.6360	0.4359	0.4013
Random	0.2931	0.6738	0.6271	0.4287	0.3994

Table 5.34: Median of the hypervolume for different values of k (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
$k = 1$	0.4180	0.4360	0.3692	0.3702
$k = 5$	0.4162	0.4380	0.3704	0.3705
$k = 10$	0.4193	0.4382	0.3704	0.3706
$k = 100$	0.4200	0.4380	0.3715	0.3706
$k = 1000$	0.4206	0.4378	0.3712	0.3705
$k = \infty$	0.4176	0.4381	0.3716	0.3706
Random	0.4067	0.4325	0.3635	0.3702

median of the hypervolume obtained at the end of the executions are shown in Tables 5.33, and 5.34. Independently of the parameterisation, the model HV_WEIGHT has reported better results than an assignment based on the RANDOM approach. It is also clear that the parameterisation $k = 1$ is not adequate. It can be observed than in most of the cases such a value reports worse solutions than the rest of the parameterisations. The cause is that, when $k = 1$ is used, the stochastic behaviour of the low-level configurations provokes fails in the way of assigning the resources. In the cases where different configurations are best suited for different stages of the optimisation, the models that use intermediate values have the ability of detecting it faster than the models that use a global adaptation level. However, in some cases the stochastic behaviour might produce some drawbacks. Thus, in some cases the global adaptation level is the best one, while in other cases intermediate values are better.

The scheme with a global adaptation level was statistically compared with the remaining ones. Tables 5.35 and 5.36 show the results of the comparison. In each

Table 5.35: Statistical comparison between $k = \infty$ and other values of k (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
$k = 1$	$\leftrightarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$k = 5$	$\downarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$
$k = 10$	$\leftrightarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
$k = 100$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
$K = 1000$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$

Table 5.36: Statistical comparison between $k = \infty$ and other values of k (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
$k = 1$	$\leftrightarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$k = 5$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$
$k = 10$	$\downarrow$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$
$k = 100$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
$k = 1000$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$	$\leftrightarrow$

Table 5.37: Factor of additional evaluations performed with each k respect to $k = \infty$ (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
$k = 1$	1.16	1.35	2.23	2.06	1.89
$k = 5$	0.88	0.78	0.88	0.89	1.02
$k = 10$	1.08	0.82	0.97	1.02	0.91
$k = 100$	1.03	0.94	1.06	1.04	0.93
$k = 1000$	1.03	1.17	1.06	1.19	0.90

cell a $\uparrow$ is shown if the model with a global adaptation level is better than a model with the corresponding value of k (shown in the first column). The statistical tests confirms that the value $k = 1$ is not adequate. The remaining values have been in some cases better than the model with $k = \infty$, while in other cases they have been statistically worse.

Finally, it is important to quantify the differences among them. The factors between the evaluations required to obtain a 50% of success ratio with HV_WEIGHT

Table 5.38: Factor of additional evaluations performed with each k respect to $k = \infty$ (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
$k = 1$	0.81	2.00	1.67	1.08
$k = 5$	1.09	1.05	1.40	1.00
$k = 10$	0.68	0.96	1.32	0.82
$k = 100$	0.58	1.06	1.00	0.76
$k = 1000$	0.54	1.17	1.08	1.18

considering different values of k, and with the HV_WEIGHT considering $k = \infty$ have been calculated. Thus, a value greater than 1 means that the model with global adaptation level converged faster than a model with the corresponding value of k. The factors are shown in tables 5.37 and 5.38. In general the values are not very large nor very small. This means that the model is robust with respect to the value of k. The exception is the case with $k = 1$, where some large values have been found. The reasons of the inadequate performance of such a parameterisation have been previously described.

Fourth Experiment: Probabilistic vs. Elitist Selection

It is also interesting to compare the previous approach, with the ELI-HV_WEIGHT scheme. ELI-HV_WEIGHT was executed with the different values of k considered in the previous experiments and with different amount of low-level configurations. In the case of using few low-level configurations (less than 16), differences among the approaches were not significant. The resources were assigned in very similar ways. However, when a higher amount of low-level configurations were used, some statistically significant differences appeared. Generally, ELI-HV_WEIGHT obtained better values than HV_WEIGHT. The reason is that HV_WEIGHT tends to distribute the resources among more configurations, while ELI-HV_WEIGHT tends to focus on few configurations. This can be appreciated in Figure 5.25. It shows the number of evaluations executed by the configuration that was granted with the maximum amount of resources. It can be appreciated that the number of evaluations executed in the case of the ELI-HV_WEIGHT model is generally higher. In some cases, such type of assignment clearly provides advantages. For instance, in the WFG1 problem ELI-HV_WEIGHT model obtains higher hypervolume values than HV_WEIGHT and the differences are statistically significant. However, in other cases, the hyper-heuristic is confounded and assigns a high number of resources among non-suitable

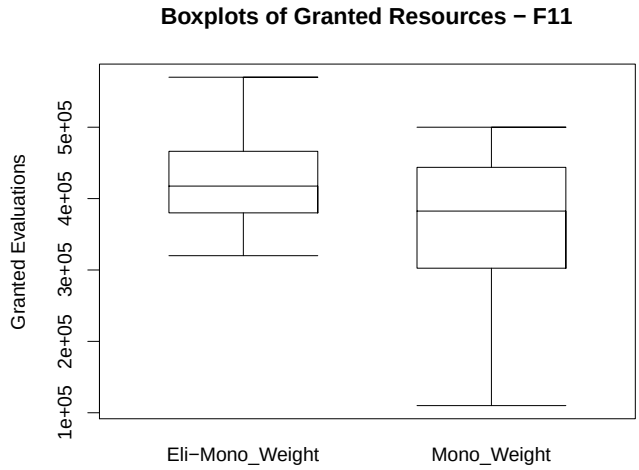


Figure 5.25: Boxplots of the resources granted to the configuration that executed more evaluations

configurations. This happens in some executions with WFG6. In such cases, the ELI-HV_WEIGHT assigns a higher amount of resources to non-suitable configurations. In fact, with WFG6 and $k = 10$, the HV_WEIGHT obtains higher hypervolume values than ELI-HV_WEIGHT and the differences are statistically significant. By exploring each execution performed with ELI-HV_WEIGHT it could be checked that in some executions the resources had been granted to non-adequate low-level approaches.

Fifth Experiment: Performance of the parallel hyperheuristic

This section is devoted to present the experiments that have been performed with the aim of validating the adequate performance of the parallel version of the approach. Specifically, a hyperheuristic that considered 16 configurations - the same as in the previous study - was executed in parallel. The same parameterisation as in the case of the first sequential experiment was used. Specifically, the value of k was fixed to ∞, and the value of β was fixed in a way that the 10% of the decisions performed by the hyperheuristic followed a uniform distribution, i.e., $\beta * n_h = 0.1$. Finally, the local stopping criterion was fixed to 5.000 function evaluations and the global stopping criterion to 1.000.000 function evaluations. Thus, the only change was the specification of the number of processors to use.
Since it is known that the migration stage highly affects the performance of island-

Table 5.39: Speedup of the parallel approach (4 islands) with HV-WEIGHT and different migration stages (WFG1 - WFG5)

	WFG1	WFG2	WFG3	WFG4	WFG5
RANDOM-NSGA-II_CROWD-ALL_ALL	3.12	3.63	4.73	4.76	5.17
ELI_RAND-NSGA-II_CROWD-ALL_ALL	4.00	3.26	3.06	3.50	7.04
RANDOM-ELI100-ALL_ALL	2.20	3.24	4.18	2.44	2.62
ELI_RAND-ELI100-ALL_ALL	2.16	3.04	3.91	2.26	2.04
RANDOM-RANDOM-ALL_ALL	2.4	3.30	3.16	2.95	3.91
ELI_RAND-RANDOM-ALL_ALL	2.44	3.20	3.12	2.97	3.06
RANDOM-NSGA-II_CROWD-RING	2.52	3.18	3.34	3.34	4.00
ELI_RAND-NSGA-II_CROWD-RING	3.36	2.81	3.10	3.50	6.06
RANDOM-ELI100-RING	1.80	3.20	3.04	2.44	2.71
ELI_RAND-ELI100-RING	2.25	3.28	3.16	2.59	2.74
RANDOM-RANDOM-RING	2.65	3.20	3.54	2.77	4.09
ELI_RAND-RANDOM-RING	2.76	3.04	3.10	3.11	3.04

based models, the aim of the first analysis has been to measure the impact that the migration stage has over the performance. The hyperheuristic was executed with 4 processors (n_p) considering several types of migration stages. Specifically, 12 different migrations stages were considered. They were made up by combining two migration selectors, three exchange selectors and two topologies. The migration selectors were the RANDOM, and the ELI_RAND. The considered exchange selectors were the RANDOM, the ELI100, and the NSGA-II_CROWD. Finally, the tested topologies were the ALL and the RING. In order to identify the different migration stages the following nomenclature has been used: *Migration Scheme-Replacement Scheme-Topology*. For example, the migration stage which uses the elitist random migration selector, with the elitist 100% ranking exchange scheme, and considers a ring topology is named as ELI_RAND-ELI100-RING.

In order to measure the performance of the parallel approach, the sequential version of HV_WEIGHT that considered 16 configurations was used to perform the comparison. Tables 5.39 and 5.40 show the speedup factors obtained with the different models for each one of the WFG problems. The factors have been calculated using the evaluations required by each model to achieve a 50% of success ratio. The speedup factors have been calculated as in the study performed for the mono-objective hyperheuristics. The calculated speedup factors reveal that the obtained performance

Table 5.40: Speedup of the parallel approach (4 islands) with HV-WEIGHT and different migration stages (WFG6 - WFG9)

	WFG6	WFG7	WFG8	WFG9
RANDOM-NSGA-II_CROWD-ALL_ALL	5.79	3.55	5.41	9.40
ELI_RAND-NSGA-II_CROWD-ALL_ALL	7.00	3.91	5.25	10.44
RANDOM-ELI100-ALL_ALL	4.09	2.26	2.81	2.50
ELI_RAND-ELI100-ALL_ALL	4.94	2.04	3.39	2.50
RANDOM-RANDOM-ALL_ALL	3.36	2.66	2.93	4.08
ELI_RAND-RANDOM-ALL_ALL	2.76	2.79	3.64	4.70
RANDOM-NSGA-II_CROWD-RING	3.87	3.25	3.68	6.26
ELI_RAND-NSGA-II_CROWD-RING	3.34	3.82	6.34	9.40
RANDOM-ELI100-RING	3.16	2.12	3.27	4.94
ELI_RAND-ELI100-RING	7.00	2.12	3.90	5.22
RANDOM-RANDOM-RING	4.54	2.69	3.31	4.47
ELI_RAND-RANDOM-RING	2.85	2.79	2.97	3.27

highly depends on the migration stage. The most important components have been the topology and the exchange selector, while the migration selector has not been so important. The migration stages ELI_RAND-NSGA-II_CROWD-ALL and RAND-NSGA-II_CROWD-ALL seem the most adequate ones. They have obtained good speedup factors in every addressed problem.

A second experiment with the aim of analysing the scalability of the approach has been performed. The same model as in the previous experiment was executed with up to 32 processors. The two best migration stages of the previous experiment were considered. Figures 5.26 and 5.27 show the speedup factors of the approach with the different optimisation problems. It can be observed that in several cases superlinear speedups have appeared. However, in other cases the obtained factors have been much lower. Anyway, in any of the cases as more processors have been added, the speedup factors have increased. The most outstanding case with 32 processors appeared with WFG6. In such a case the speedup factor was 33.6. The problem that produced the worst case with 32 processors was WFG1. The obtained speedup factor was 11.71.

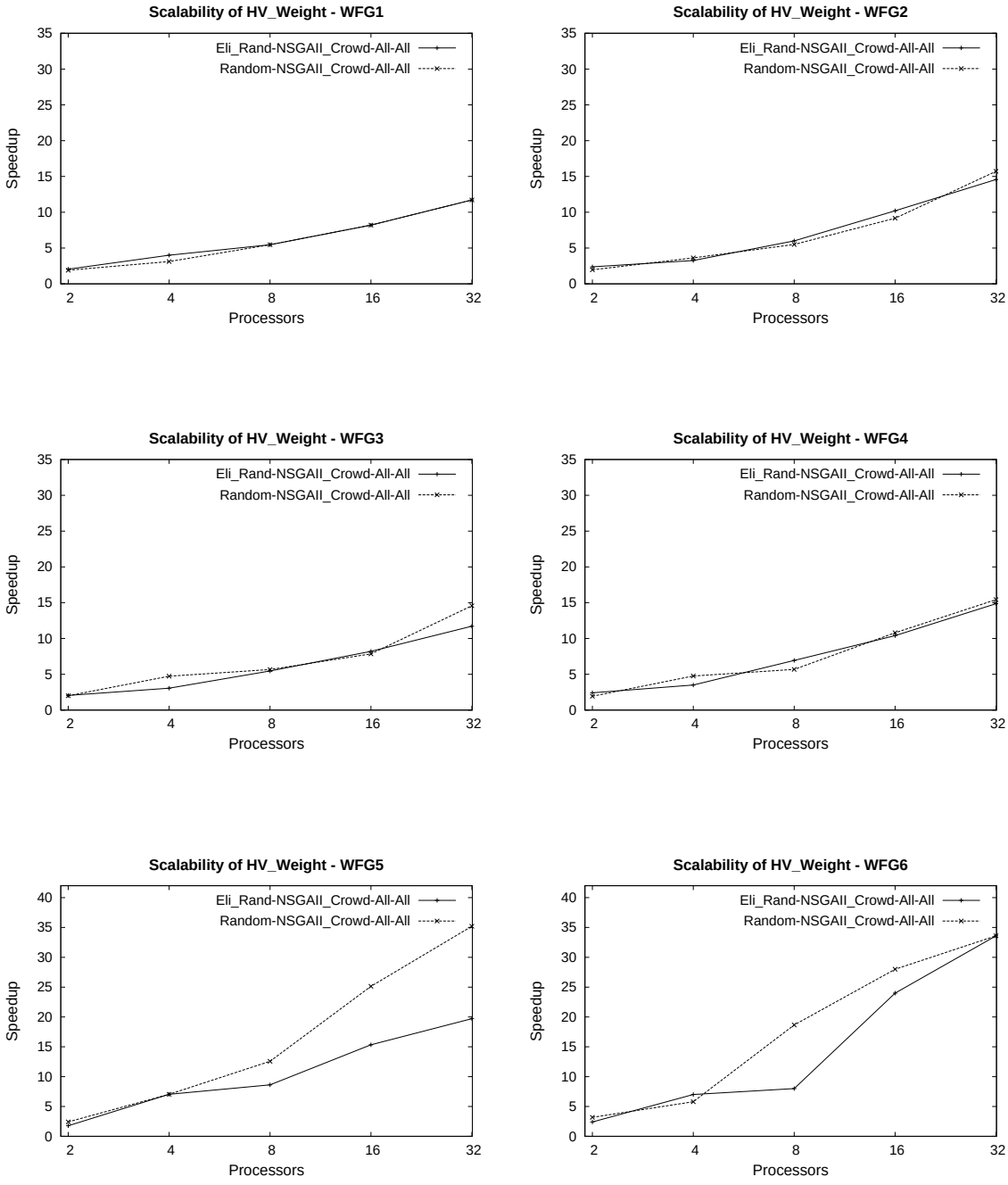


Figure 5.26: Scalability analysis for different migration stages (WFG1 - WFG6)

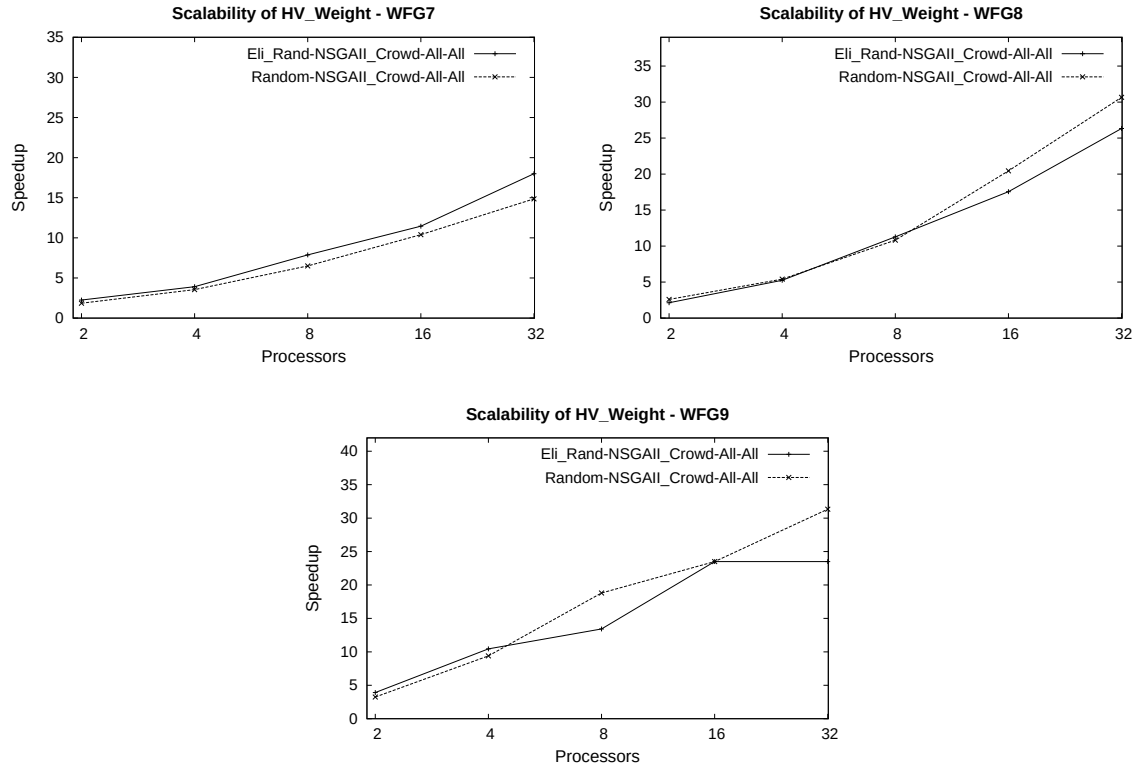


Figure 5.27: Scalability analysis for different migration stages (WFG7 - WFG9)

Final Remarks

Additional experiments with other low-level configurations have been carried out. Specifically, several experiments with the algorithms SPEA2 and IBEA have been performed. Additionally, alternative crossover and mutation operators have been used. Specifically, the polynomial mutation, and the uniform crossover have been considered. Such experiments have confirmed the adequate performance of the hyperheuristic. The obtained results differ in quality and velocity of convergence. However, regarding the usage of the analysed hyperheuristic, similar conclusions than the ones obtained with the previous analysis can be drawn.

Part III

Practical Applications

Chapter
6

Communication Optimisation Problems

This chapter is devoted to present a set of problem-dependent techniques for several optimisation problems that arise in the telecommunication field. Such techniques have been merged with the problem-independent schemes previously presented. Two of the problems arise in the design of wireless networks. The last addressed problem deals with the design of an optimal broadcasting strategy for Mobile Ad-Hoc Networks.

6.1 Antenna Positioning Problem

6.1.1 General Problem Description

Engineering of wireless telecommunication networks evolves two major optimisation problems [178]: the Antenna Positioning Problem (APP) and the Frequency Assignment Problem (FAP). This section focuses on the APP. The aim of APP is to position a set of base stations (BS) or antennas on potential sites, in order to fulfil some objectives and constraints. Several objectives can be considered when designing a network. Most typical considered objectives are: minimise the number of antennas, maximise the amount of traffic held by the network, maximise the quality of service, and/or maximise the covered area. The APP has been analysed by many researchers and it has been shown to be an NP-Complete optimisation problem [109]. APP plays a major role in various engineering, industrial, and scientific applications because its outcome usually affects cost, profit, and other heavy-impact business performance metrics. Therefore, the quality of the applied approaches has a direct bearing on industry economic plans.
Several mathematical formulations of the APP have been proposed [6, 232]. In

some cases, the models are based on using a network simulator that incorporates a wave propagation model to estimate the performance of a given solution. Among others, the free space, the Hokumara-Hata and the Walfish-Ikegami models can be used [214]. In other cases, the canonical APP is used [246]. The main benefit of the canonical version is that the formulation is independent of the used technology. Therefore, new instances can be tackled with little effort. The technological constraints would raise the combinatorial complexity of the problem. However, the problem's essence remains untouched. Thus, using such a formulation is an inexpensive way of comparing different algorithms for the APP.

6.1.2 Mathematical Formulation

APP is defined as the problem of identifying the infrastructures required to establish a wireless network. The APP mathematical formulation used in this research was proposed in [246]. Such a formulation comprises the maximisation of the coverage of a given geographical area while minimising the amount of base stations deployed. Thus, it is an intrinsically multi-objective problem. A BS is a radio signal transmitting device that irradiates any type of wave model. The region of the area covered by a BS is called a cell. In our definition of APP, BSs can only be located in a set of potential locations. The formulation considers two objectives: the maximisation of the coverage ($Coverage$), and the minimisation of the BSs or transmitters deployed ($Transmitters$). In [6, 246] the problem is simplified by defining a fitness function that converts the problem into a mono-objective one.
The geographical area G on which a network is deployed is discretised into a finite number of points or locations. Tam_x and Tam_y are the number of vertical and horizontal subdivisions, respectively. They are selected by communications experts, depending on several characteristics of the region and transmitters. U is the set of locations where BSs can be deployed: $U = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$. Location i is referred to using the notation $U[i]$. The x and y coordinates of location i are named $U[i]_x$ and $U[i]_y$, respectively. If a BS is located in position i, its corresponding cell is covered. The cell is named $C[i]$. In our definition we use the canonical APP formulation, i.e. an isotropic radiating model is considered for the cell. The set P determines the locations covered by a BS: $P = \{(\Delta x_1, \Delta y_1), (\Delta x_2, \Delta y_2), ..., (\Delta x_m, \Delta y_m)\}$. Thus, if BS i is deployed, the covered locations are given by the next set: $C[i] = \{(U[i]_x + \Delta x_1, U[i]_y + \Delta y_1), (U[i]_x + \Delta x_2, U[i]_y + \Delta y_2), ..., (U[i]_x + \Delta x_m, U[i]_y + \Delta y_m)\}$. Being $B = [b_0, b_1, ..., b_n]$ the binary vector that determines the deployed BSs, the next two objectives are defined:
$f_1 = \sum_{i=0}^{n} b_i$
$f_2 = \sum_{i=0}^{tam_x} \sum_{j=0}^{tam_y} covered(i, j)$

where:

$$covered(x,y) = \begin{cases} 1 & If\ \exists\ i/\{(b_i\ =\ 1)\ \wedge\ ((x,y) \in C[i])\} \\ 0 & Otherwise \end{cases} \quad (6.1)$$

Objective f_1 ($Transmitters$) is the number of deployed BSs, so it must be minimised. Objective f_2 ($Coverage$) is a measure of the covered area, so it must be maximised. In the cases where the problems is converted to a mono-objective optimisation problem, the following fitness function must be maximised:

$$f(solution) = \frac{Coverage^{\alpha}}{Transmitters} \quad (6.2)$$

In the previous scheme a decision maker must select a value for α. It is tuned considering the importance given to the coverage, in relation to the number of deployed BSs.

6.1.3 Proposed Optimisation Schemes

Mono-objective Metaheuristics

First, a set of mono-objective metaheuristics were designed and tested with a real-world sized instance. A large effort was done to adapt the metaheuristics to the problem at hand, creating new operators and several hybrid schemes. The analysis was done in collaboration with researchers of several universities. Thirteen metaheuristics were tested. They were: SA, CHC, ILS, PBIL, DE, GRASP, VNS, and several hybrid approaches. The main duty of my research group was to design and implement a version of ILS. Thus, the details of such an implementation are given. The basic behaviour of the rest of the approaches were given in Chapter 2. The details for such algorithms can be found in [176].
The general pseudocode of ILS has been previously presented. The parts which are specific for the APP are the following:

- A method for generating initial solutions.
- A method for perturbing the solutions.
- The neighbourhood definition.

In order to generate the initial solutions the following steps are executed. First, the grid is divided into a set of sub-grids or windows. All windows have size $N \times N$, where N is randomly selected between the values in the range $[(R-9) \times 2, R*2]$. R represents the antennas coverage radius. The centre of each window is calculated and each coordinate is shifted considering random values in the range $[-5, 5]$. The antenna which is nearest to each calculated position is inserted in the current solution.
The designed perturbation mechanism selects a set of deployed transmitters to be removed from the solution and a set of locations in which to include an extra antenna. For perturbing a solution, the number of antennas to be deleted and inserted are determined by a random value that follows a normal distribution of mean $m * strength$ and standard deviation sd. The BSs to be discarded from the solution and the ones to be included are randomly selected from the set of available locations. Once such modifications have been introduced into the solution, a final correction is performed. For each base station location, the fitness of the solution including the transmitter (if it is not used in the current solution) or excluding the transmitter (if it is used in the current solution) is checked and the best choice is selected for the final solution. Moreover, iterated local search controls the strength of the perturbation. Initially it is set to 1. If the best solution has not been improved in the last b search iterations, the search strength is increased. Moreover, if after i increases the search keeps trapped in a local optimum, the algorithm is restarted from a new generated initial solution.
The intensification step is performed with a hill climbing local search. The neighbourhood of a solution is defined as follows:

1. For each of the available locations that have not been used in the solution, a neighbour that includes an antenna in the corresponding position is created.

2. For each of the available locations where a transmitter has been placed, a neighbour that excludes such an antenna from the solution is created.

3. For each of the available locations where a transmitter has been placed, a neighbour that replaces such a base station with the nearest one is created.

During the local search, the complete neighbourhood is generated. From the neighbours obtained, the best one is selected. The process finishes when the local search reaches a local maximum or when the steps are repeated a maximum number of ms times. Finally, for the acceptance criterion of the ILS, the solution with greatest fitness is always selected.

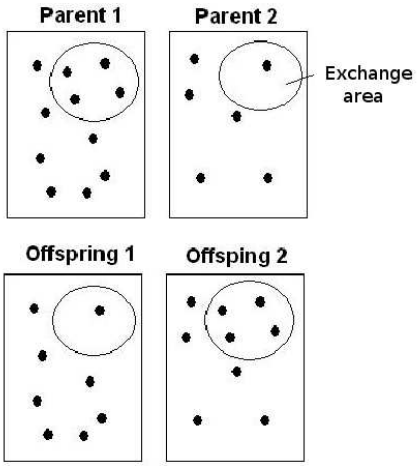


Figure 6.1: Creation of offspring with GC

Multi-objective Metaheuristics

One of the main drawbacks of the previous approaches is that the practitioner must select a value for the α parameter of the fitness function. For this reason, additional metaheuristics that deals with the multi-objective version of the problem have been proposed. In addition, they were integrated with the HV_WEIGHT hyperheuristic and executed in a parallel environment. Results obtained with the mono-objective approaches have been used to validate the correct behaviour of the multi-objective schemes.

The metaheuristics tested for the multi-objective version of the APP have been four MOEAs: NSGA-II, SPEA2, and IBEA (both the adaptive and non-adaptive version). In order to apply the previous MOEAs, an encoding for the individuals must be defined. Tentative solutions have been represented as binary strings with n elements. Each gene determines whether the corresponding BS is deployed or not.

Also, a set of variation operators must be defined in order to employ such MOEAs. The mutation operator was the Flip operator. Each gene is inverted with a probability p_m. Two different crossover operators - one random, and one directed - were tested. The crossover operators were the OPX, and a geographic crossover [232]. The geographic crossover (GC) is illustrated in Figure 6.1. First, a random location is chosen. Then, the parents exchange the BSs that are located within a given radius (r) around the selected location.

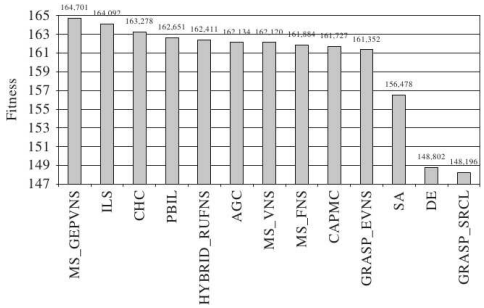


Figure 6.2: Mean of the fitness obtained at the end of the executions

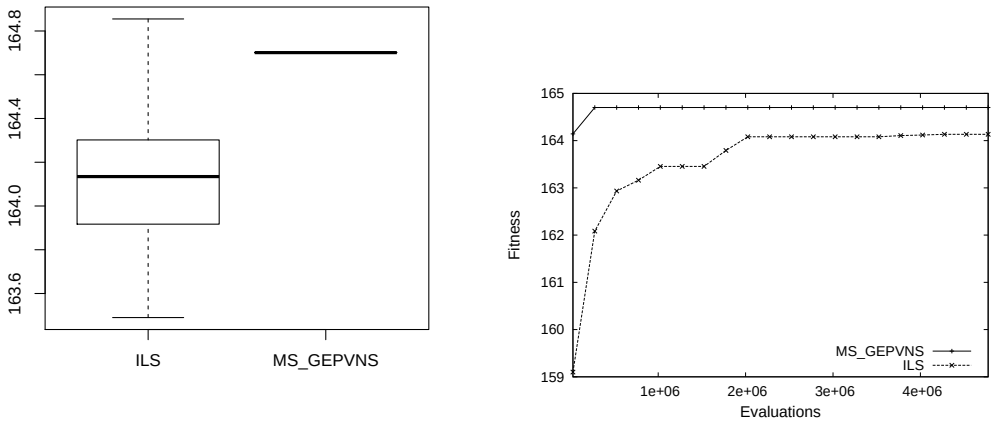
6.1.4 Experimental Evaluation

Mono-objective Metaheuristics

The experimental evaluation of the mono-objective metaheuristic was performed with a real-world-sized problem instance, defined by the geographical layout of the city of Malaga. Such an instance had been previously analysed by some communication experts [6], who had determined that the value $\alpha = 2$ was adequate. This instance represents an urban area of $27.2km^2$. The terrain has been modelled using a 450 x 300 grid, where each point represents a surface of approximately 15 x 15 m. A dataset containing 1.000 candidate sites for the BSs is used. The cell model for the BSs coverage is an omnidirectional isotropic model, with a radius of approximately one half kilometre (30 grid points).

Every experiment in this chapter has been run on a Debian GNU/Linux cluster of 8 HP nodes, each one consisting of two Intel(R) Xeon(TM) at 3.20GHz and 1Gb RAM. The compiler and MPI implementation used were *gcc 3.3* and MPICH *1.2.7*. The interconnection network has been a Gigabit Ethernet.

For each metaheuristic some preliminary experiments were performed with the aim of setting up the different parameters. In the case of ILS, it was executed considering the following parameterisation: $m = 3$, $sd = 1$, $ms = 100$, $b = 2500$, $i = 2$. The metaheuristics were executed considering a stopping criterion of 5.000.000 function evaluations. Figure 6.2 shows the mean of the fitness obtained by the different tested metaheuristics at the end of the executions. The obtained results demonstrate the adequate performance of the ILS approach. In fact, ILS has obtained the second best mean. In order to better inspect the obtained results, Figure 6.3 show the evolution of the median of the fitness obtained by the best algorithms. The boxplots obtained at the end of the executions are also shown. It can be observed that



(a) Boxplots obtained in 5.000.000 function evaluations

(b) Evolution of the mean fitness

Figure 6.3: Fitness obtained by the best mono-objective approaches

ILS has been the algorithm that has obtained the best overall solution. The main drawback of MS_GEPVNS - the approach that obtained the highest mean - is that it has reached in every case the same local optimum. The solution has a high fitness, but independently of the invested time, the approach never escapes from such a solution. In fact, it obtains such a solution in only 250.000 evaluations. However, ILS has been able to improve on such a solution in several executions. Thus, in order to obtain high-quality solution in few evaluations, the usage of MS_GEPVNS is preferred. However, if many resources are going to be used, the application of ILS might provide better solutions.

Multi-objective metaheuristics

The experimental evaluation performed for the multi-objective metaheuristics has considered the same instance than in the mono-objective case. In our first experiment a comparison among the different MOEAs was carried out. The comparisons are based on the obtained hypervolume. Each MOEA was executed with a stopping criterion of 2 hours. For each MOEA, 15 parameterisations were analysed. They were made up by combining the flip mutation operator with $p_m = \{0.001, 0.002, 0.004\}$ and the OPX operator with $p_c = \{0, 0.25, 0.5, 0.75, 1\}$. Table 6.1 shows the number of column configurations which are worse (significant

Configuration	SPEA2	Adapt. IBEA	IBEA	NSGA-II
SPEA2	0	15	15	15
Adapt. IBEA	0	0	14	11
IBEA	0	0	0	10
NSGA-II	0	3	4	0

Table 6.1: Statistical comparison of MOEAs

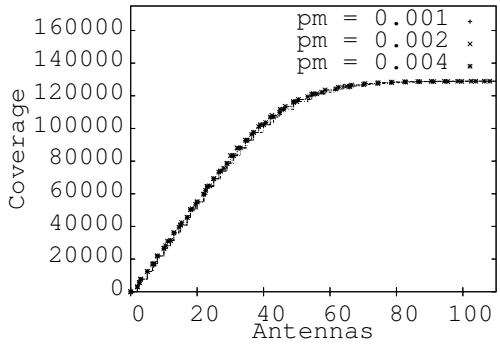


Figure 6.4: Attainment Surfaces with several p_m values

differences) than the corresponding row configuration. For the analysed instance, SPEA2 is clearly the best strategy in terms of the achieved hypervolume. Results show that SPEA2 is better than any other algorithm, with any of the tested parameterisations.

The variation process in MOEAs is performed by combining mutation and crossover operators. The probability of mutation is usually fixed as $1/n$, being n the number of genes. The next experiment tests the robustness of the mutation operator. Figure 6.4 shows the 50%-attainment surfaces for SPEA2 with the flip mutation operator using $p_m = \{0.001, 0.002, 0.004\}$, and $p_c = 0$. We can note that changing p_m in the selected range do not produce a high effect in the achieved results.

On the other hand, according to [232], it is necessary to include problem-dependent information in the crossover, in order to develop a non-destructive operator. Figure 6.5 shows the 50%-attainment surfaces for SPEA2, with $p_c = \{0, 1\}$, and $p_m = 0.001$. It confirms that OPX is not providing any improvement in the achieved results.

Next experiment tested the advantages of including problem-dependent information in the crossover operator. Operator GC was applied with different crossover probabilities, and compared with the OPX with $p_c = 1$. Table 6.2 shows whether the

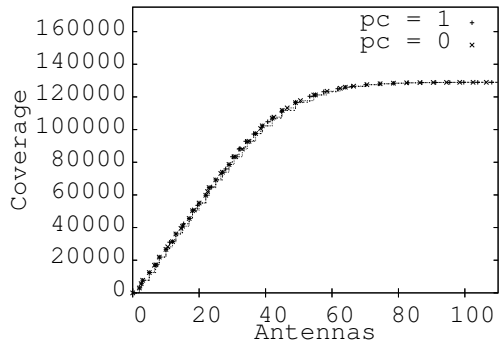


Figure 6.5: Attainment Surfaces with several p_c values

CONFIGURATION	OPX, $p_c = 1$	GC, $p_c = 0.25$	GC, $p_c = 0.5$	GC, $p_c = 0.75$	GC, $p_c = 1$
OPX, $p_c = 1$	$\leftrightarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
GC, $p_c = 0.25$	$\uparrow$	$\leftrightarrow$	$\leftrightarrow$	$\downarrow$	$\downarrow$
GC, $p_c = 0.5$	$\uparrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\downarrow$
GC, $p_c = 0.75$	$\uparrow$	$\uparrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
GC, $p_c = 1$	$\uparrow$	$\uparrow$	$\uparrow$	$\leftrightarrow$	$\leftrightarrow$

Table 6.2: Advantages of incorporating problem-dependent information

row configuration is statistically better ($\uparrow$), not different ($\leftrightarrow$), or worse($\downarrow$), than the corresponding column configuration. It shows the benefits of incorporating problem-dependent information into the operator. Moreover, the best results have been attained with high values of p_c.

The GC uses a parameter which represents the radius (r) of the exchanged area. It is interesting to test the behaviour of such an operator with different values of r. SPEA2 was executed with GC fixing the next values for the radius: $r = \{15, 30, 45, 60\}$. Table 6.3 shows the statistical comparison with the same meaning as table 6.2. The best behaviour is obtained by using the parameterisation $r = 30$. This makes sense because, since the antennas radius is also 30, the exchanged area coincides with the area influenced by the randomly selected antenna.

Finally, Figure 6.6 compares the results obtained by multi-objective and mono-objective approaches. It shows the 50%-attainment surface achieved with the best tested configuration of SPEA2, as well as the best solution obtained by the best known mono-objective strategy. On one hand, the multi-objective schemes are obtaining many solutions which are non-dominated by the best mono-objective solutions (in average the 97% of the solutions are non-dominated). Thus, the schemes

Configuration	GC, $r = 15$	GC, $r = 30$	GC, $r = 45$	GC, $r = 60$
GC, $r = 15$	$\leftrightarrow$	$\downarrow$	$\downarrow$	$\downarrow$
GC, $r = 30$	$\uparrow$	$\leftrightarrow$	$\uparrow$	$\uparrow$
GC, $r = 45$	$\uparrow$	$\downarrow$	$\leftrightarrow$	$\uparrow$
GC, $r = 60$	$\uparrow$	$\downarrow$	$\downarrow$	$\leftrightarrow$

Table 6.3: Statistical comparison of GC with different radius

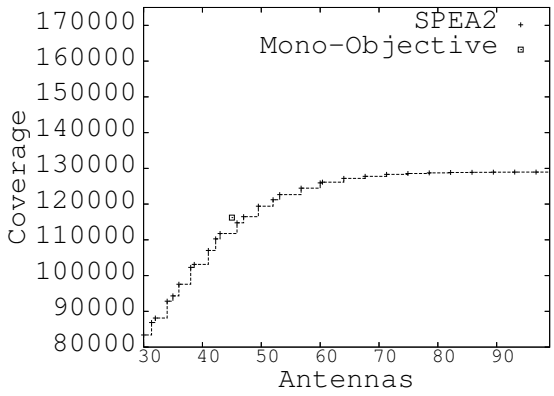
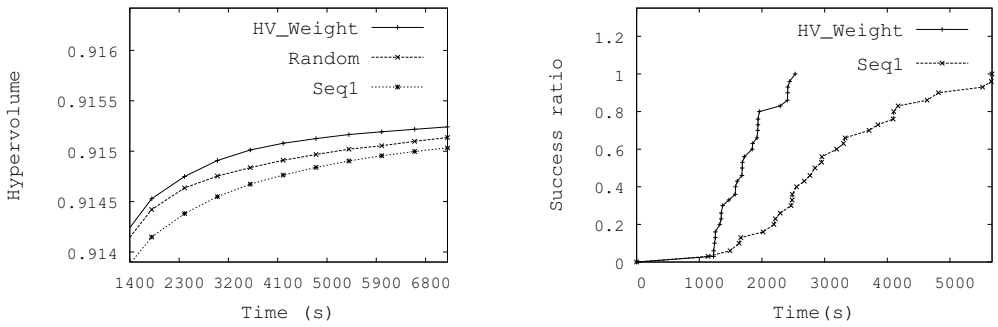


Figure 6.6: Comparison of multi-objective and mono-objective approaches

are providing a large number of high-quality solutions. On the other hand, since the mono-objective solution dominates about 3% of the achieved solutions, there is still some room for improvement for the multi-objective schemes.

Finally, the performance of the hyperheuristic HV_WEIGHT, executed in a parallel environment, has also been analysed. The configurations were made up by combining different parameterisations of mutation and crossover operators. Every configuration used the SPEA2. The flip mutation operator was used with $p_m = \{0.001, 0.002, 0.004\}$. The set of crossover operators was the following: OPX with $p_c = \{0.5, 1\}$, and GC with $(p_c, r) = \{(0.5, 15), (0.5, 30), (0.5, 45), (0.5, 60), (1, 15), (1, 30), (1, 45), (1, 60)\}$. Thus, a set of 30 low-level configurations was considered. The population and archive sizes were fixed to 100. Each considered scheme was executed fixing a stopping criterion of 2 hours. Considering the obtained results, sequential algorithms were ordered based on the mean hypervolume achieved at the end of the executions. An index based on such an order is assigned to each configuration, being "Seq1" the one obtaining the highest hypervolume.

The parallel model was executed using HV_WEIGHT with the 30 described configurations. The executions used 4 worker islands. The following parameterisation was



(a) Evolution of the mean hypervolume (b) Run-length Distribution

Figure 6.7: Comparison of HV_WEIGHT and Seq1

used: $\beta = 0.2/30$ and $k = 20$. The local stopping criterion was fixed to 1 minute. In addition, a random parallel scheme was used. In such a scheme, every configuration is scored with the same value. Migration was performed following an asynchronous scheme with a migration probability of 1, but it only takes place when new non-dominated individuals have been generated. The ALL topology was considered. Migrated individuals are selected following the ELI-RAND scheme. Replacements were performed following the ELI100 selector.

Figure 6.7-(a) shows the hypervolume evolution for "Seq1" and for the parallel models "random" and HV_WEIGHT. It shows that every parallel model is better than the best sequential configuration. Therefore, the parallel models are useful, not only to avoid the testing of each sequential configuration, but also, to speed up the attainment of high-quality solutions. Moreover, the results of HV_WEIGHT clearly improve on the ones achieved by the "random" model. A statistical comparison was performed among the models. It shows that the parallel models are better than the sequential configuration. In addition, it confirms that the differences between HV_WEIGHT and the random approach are significant.

The previous experiment has compared the schemes, mainly focused in terms of the achieved hypervolume. However, since the parallel executions use more computational resources than the sequential ones, the improvement must be quantified. In order to measure the improvement of the parallel approach the RLDs have been used. The HV_WEIGHT and the "Seq1" model were executed using as finalisation condition the achievement of a fixed level of hypervolume. Such hypervolume was the mean of the hypervolume attained by "Seq5". Figure 6.7-(b) shows the RLDs for "Seq1" and HV_WEIGHT. It shows the superiority of the HV_WEIGHT model. The speedup factor for achieving a 50% of success ratio is 1.71. Moreover, the

speedup factor is much larger when other sequential configurations are considered. For instance, when "seq5" is used as the reference, the speedup factor is 5.15. This mean, that, by using four processors the practitioner might avoid the testing of each low-level configurations, and speedup the achievement of high-quality solutions.

6.2 Frequency Assignment Problem

6.2.1 General Problem Description

Wireless communication networks have undergone a dramatic expansion over the last few decades. The Frequency Assignment Problem (FAP) plays a key role in the design of these kinds of networks [1]. Several variants of the FAP have emerged for the various wireless technologies. Specifically, in the case of the GSM networks, the FAP is one of the crucial issues to be considered in its design [187]. This problem is also known as *Automatic Frequency Planning* (AFP) and *Channel Assignment Problem* (CAP). Although the FAP has led to many different mathematical and engineering models, all of them share two common features:

- A set of antennas must be assigned frequencies such that data transmissions between the two end points of each connection are possible.
- Depending on the frequencies assigned to the antennas, they may interfere with one another, resulting in loss of signal quality.

The performed research focuses on the FAP that arises in the design of GSM networks. In such a case, the available frequency band is slotted into channels that have to be allocated to the elementary transceivers (TRXs) installed in the base stations of the network. In GSM, the FAP is a hard design task because the usable radio spectrum is very scarce and frequencies have to be reused throughout the network; consequently, some degree of interference is inevitable. The goal of the designer is to minimise the interferences in the network, i.e. to minimise the loss of signal quality. The FAP emerges in different network environments, and involves different objectives, features, and constraints. Therefore, several mathematical formulations have been defined for dealing with the FAP [1]. In the last few years, the basic FAP formulation has been widely extended in order to address real-world issues [149]. Most of the FAP models differ in the way that the interference is measured. Computing the level of interference is a difficult task which depends on the channels, the radio signals and many other features of the environment. The quantification of the interference results in an *interference matrix*, usually denoted by M. Some theoretical methods for measuring M have been proposed [1]. Theoretical methods offer the

advantage of allowing for new instances to be tackled with less effort. However, these methods ignore some features of the environment, making it difficult to know the consequences that its usage might involve. Therefore, in other researches [149], extensive network measurements are performed in order to calculate M. In such cases, the M matrix is composed of more accurate values, resulting in more realistic frequency plans. However, applying the method to new networks is expensive.

The FAP can be classified in different ways depending on the spectrum size, the objectives, and the specific technological constraints. In [1], three main FAP models are described: Minimum Order Frequency Assignment Problem (MOFAP), Minimum Span Frequency Assignment Problem (MSFAP), and Minimum Interference Frequency Assignment Problem (MIFAP). These models have chronologically appeared in the literature as technology, national regulations and markets have determined the working conditions. MOFAP is aimed at reducing the number of frequencies used in a given cellular network. It assumes that different frequencies do not interfere with each other. MSFAP focuses on searching an assignment that minimises the difference between the largest and the smallest assigned frequency, i.e. the span. It assumes that frequencies are assigned by regulators in continuous slots. Finally, MIFAP tries to minimise a measure of the overall interference in the network. MIFAP is the FAP model that has been most frequently addressed in the recent literature, mainly because of its direct applicability to the resolution of large instances of real-world GSM frequency planning [31]. Besides these three main models of the FAP, other variants are seen to exist in the literature [1].

Several optimisation methods have been used to address the different versions of the FAP. Among them, some exact algorithms have been proposed [12, 99, 170]. However, since most FAP variants are NP-hard [119], approximation approaches are mandatory when tackling large network instances [1]. Metaheuristics have been shown to yield very accurate solutions to the FAP problem [10]. Specifically, hybrid metaheuristics [230], which incorporate tailor-made local search methods, have provided very promising results [60].

The version of the FAP considered in this research [164] is classified as a MIFAP. In such a version, the interference matrix is calculated by using an extensive measurement stage. This matrix includes the interference between cells by giving the entire probability distribution of the Carrier-to-Interference ratio (C/I). The main advantage of this formulation is that it allows not only for the computation of high performance frequency plans, but also for a prediction of the Quality of Service (QoS). At the beginning of this research, the best solutions reported for this version of FAP had been reported by an ACO algorithm[164].

6.2.2 Mathematical Formulation

The FAP formulation applied here [164] is based on a matrix M calculated by extensive measurements. Let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of n *transceivers*, and let $F_i = \{f_{i1}, \ldots, f_{ik_i}\} \subset \mathbb{N}$ be the set of valid *frequencies* that can be assigned to a transceiver $t_i \in T$, $i = 1, \ldots, n$. Note that k_i - the cardinality of F_i - is not necessarily the same for all the transceivers. Furthermore, let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of given *sectors* (or cells) of cardinality m. Each transceiver $t_i \in T$ is installed in exactly one of the m sectors. Henceforth we denote the sector in which a transceiver t_i is installed by $s(t_i) \in S$. Finally, the *interference matrix* $M = \{(\mu_{ij}, \sigma_{ij})\}_{m \times m}$, is given. The two elements μ_{ij} and σ_{ij} of a matrix entry $M(i,j) = (\mu_{ij}, \sigma_{ij})$ are numerical values greater than or equal to zero. μ_{ij} represents the mean and σ_{ij} the standard deviation of a Gaussian probability distribution describing the C/I ratio [244] when sectors i and j operate on the same frequency. The higher the mean value, the lower the interference and thus the better the communication quality. Note that the interference matrix is defined at the sector (cell) level because the transceivers installed in each sector all serve the same area.
A solution to the problem is obtained by assigning to each transceiver $t_i \in T$ one of the frequencies from F_i. A solution (or frequency plan) is henceforth denoted by $p \in F_1 \times F_2 \times \cdots \times F_n$, where $p(t_i) \in F_i$ is the frequency assigned to transceiver t_i. The objective is to find a solution p that minimises the following cost function:

$$C(p) = \sum_{t \in T} \sum_{u \in T, u \neq t} C_{\text{sig}}(p, t, u)$$

In order to define the function $C_{\text{sig}}(p, t, u)$, let s_t and s_u be the sectors in which the transceivers t and u are installed; that is, $s_t = s(t)$ and $s_u = s(u)$, respectively. Moreover, let $\mu_{s_t s_u}$ and $\sigma_{s_t s_u}$ be the two elements of the corresponding matrix entry $M(s_t, s_u)$ of the interference matrix with respect to sectors s_t and s_u. Then, $C_{\text{sig}}(p, t, u) =$

$$\begin{cases} K & \text{if } s_t = s_u, |p(t) - p(u)| < 2 \\ C_{\text{co}}(\mu_{s_t s_u}, \sigma_{s_t s_u}) & \text{if } s_t \neq s_u, \mu_{s_t s_u} > 0, |p(t) - p(u)| = 0 \\ C_{\text{adj}}(\mu_{s_t s_u}, \sigma_{s_t s_u}) & \text{if } s_t \neq s_u, \mu_{s_t s_u} > 0, |p(t) - p(u)| = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The parameter K represents the cost associated with the usage of the same or adjacent frequencies in the same area. In real networks, it is unfeasible to operate with more than one transceiver with the same or adjacent frequencies serving the same area. Thus, K is defined as a very large constant. Function $C_{\text{co}}(\mu, \sigma)$ is defined

as follows:

$$C_{\text{co}}(\mu, \sigma) = 100 \left(1.0 - Q\left(\frac{c_{\text{SH}} - \mu}{\sigma}\right)\right)$$

where

$$Q(z) = \int_z^\infty \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx$$

is the tail integral of a Gaussian probability distribution function with zero mean and unit variance, and c_{SH} is a minimum quality signalling threshold. Function Q is widely used in digital communication systems because it characterises the error probability performance of digital signals [217]. This means that $Q\left(\frac{c_{\text{SH}}-\mu}{\sigma}\right)$ is the probability of the C/I ratio being greater than c_{SH}, and therefore $C_{\text{co}}(\mu_{s_t s_u}, \sigma_{s_t s_u})$ computes the probability of the C/I ratio in the serving area of sector s_t being below the quality threshold due to the interference caused by sector s_u. That is, if this probability is low, the C/I value in sector s_t is not likely to be degraded by the interfering signal coming from sector s_u, and thus the resulting communication quality is high. Note that this is compliant with the definition of a minimisation problem. In contrast, a high probability - and consequently a high cost - causes the C/I mostly to be below the minimum threshold c_{SH}, and thus results in low quality communications.
As function Q has no closed form for the integral, it has to be evaluated numerically. To do this, we use the complementary error function E:

$$Q(z) = \frac{1}{2} E\left(\frac{z}{\sqrt{2}}\right)$$

In [203], a numerical method is presented that allows the value of E to be computed with a fractional error smaller than $1.2 \cdot 10^{-7}$. Analogously, function $C_{\text{adj}}(\mu, \sigma)$ is defined as:

$$\begin{aligned} C_{\text{adj}}(\mu, \sigma) &= 100 \left(1.0 - Q\left(\tfrac{c_{\text{SH}} - c_{\text{ACR}} - \mu}{\sigma}\right)\right) \\ &= 100 \left(1.0 - \tfrac{1}{2} E\left(\tfrac{c_{\text{SH}} - c_{\text{ACR}} - \mu}{\sigma\sqrt{2}}\right)\right) \end{aligned}$$

The only difference between functions C_{co} and C_{adj} is the additional constant $c_{\text{ACR}} > 0$ (*adjacent channel rejection*) in the definition of function C_{adj}. This hardware specific constant measures the receiver's ability to receive the wanted signal in the presence of an unwanted signal in an adjacent channel. Note that the effect of constant c_{ACR} is that $C_{\text{adj}}(\mu, \sigma) < C_{\text{co}}(\mu, \sigma)$. This makes sense because using adjacent frequencies (channels) does not result in such strong interference as using the same frequency.

Algorithm 18 Local Search for the Frequency Assignment Problem

```
1: Input: current solution S
2: nextSectors ← {1, ..., numberOfSectors}
3: while (nextSectors != ∅) do
4:    currentSectors ← nextSectors
5:    nextSectors ← ∅
6:    while (currentSectors != ∅) do
7:       sec ← extract a random sector from currentSectors
8:       neighbour ← reassign frequencies of S in sector sec
9:       if (neighbour improves S) then
10:         S ← neighbour
11:         nextSectors += sectors interfered with by sec
12:         nextSectors += sectors that interfere with sec
13:      end if
14:   end while
15: end while
16: return S
```

6.2.3 Proposed Optimisation Schemes

Local Search

A local search specifically designed for this version of the FAP was designed. The application of the local search methods allows for admissible solutions to be obtained in relatively short times. Given its importance, a considerable effort was made to make the procedure as efficient as possible.

The operation of the local search (Algorithm 18) is based on optimising the assignment of frequencies to TRXs in a given sector, leaving intact the remaining network assignments. Each neighbour of a candidate solution is obtained by replacing the frequencies in the TRXs of a sector. Therefore, the neighbourhood size is the number of sectors in the network. The reassignment of frequencies within a sector is performed as follows: first, the available frequencies for the sector are sorted by their corresponding cost. Then, two possibilities are considered: either assign the frequency with the lowest associated cost to a TRX that is allowed to use that frequency, or assign its two adjacent frequencies to two different TRXs. For each of the newly generated partial solutions, the same process is repeated until all TRXs in the sector are assigned a frequency. The complete solution with the lowest associated cost is considered as the new neighbour, while the other ones are discarded.

Figure 6.8 illustrates the generation of a new neighbour. In this example, the sector

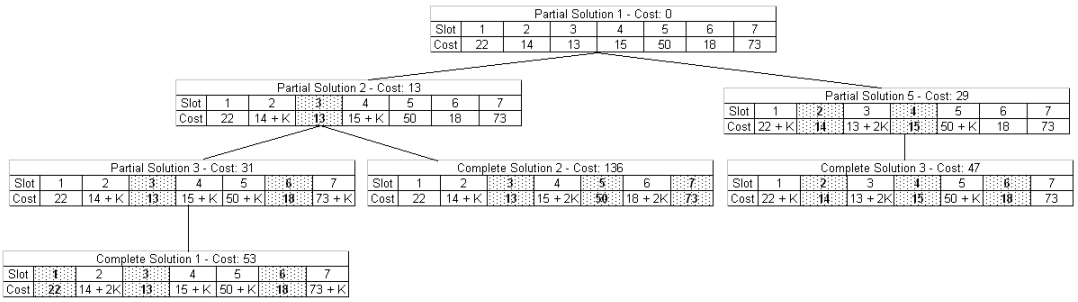


Figure 6.8: Generation of a new neighbour by reassigning the frequencies of a sector

is assumed to contain three TRXs, and that each TRX can use any frequency slot. For every node, the cost associated with each slot is shown. The children of a node are generated in accordance with the rules detailed earlier. The slots assigned to the TRXs are bolded. The nodes with three slots assigned are complete solutions, while the other ones are partial solutions. The complete solution identified by the number three is the new neighbour because it is the one with the lowest cost. The remaining generated solutions are discarded.

The order in which neighbours are analysed is randomly determined (line 7 of the Algorithm 18) while trying to avoid the generation of neighbours that do not improve on the current solution. To this end, a set called *currentSectors* containing the sectors that might improve on the current solution is stored. Initially, all sectors are introduced in *currentSector* (lines 2 and 4). For the generation of a new neighbour, a sector *sec* is randomly extracted from *currentSector* (line 7) and its frequencies reassigned as discussed above (line 8). The local search moves to the first new generated neighbour that improves on the current solution (lines 9-10), adding all the sectors that interfere with or are interfered with by *sec* to the set of the next sectors (*nextSectors*) to be considered (lines 11-12). When the *currentSectors* set becomes empty (line 6), sectors in *nextSectors* are transferred to the current set (line 4) and the *nextSectors* set is cleared (line 5). The local search stops when none of the neighbours improves on the current solution (line 3).
In cases where the network satisfies a set of properties, the neighbour generation process ensures the achievement of the optimal frequency assignment inside the analysed sector, considering the remaining network fixed. Such properties are (i) all TRXs in a given sector are allowed to use the same frequency ranges, (ii) it is possible to make assignments which do not use the same frequency or adjacent frequencies in any two TRXs serving the same area, and (iii) the optimal assignment

does not use the same frequency or adjacent frequencies in any two TRXs that are in the same sector. A sketch of the proof is here presented. Let $Cost(f)$ be the cost associated to the assignment of the frequency f to any of the TRXs in the considered sector. Being f_1 the frequency with minimum associated cost, the best assignment must use f_1, or must simultaneously use $f_1 - 1$ and $f_1 + 1$. In fact, considering an assignment in which $f_1 + 1$ is used, but $f_1 - 1$ is not used, we can substitute the assignment of $f_1 + 1$ by f_1, thus obtaining an assignment with lower cost. In the case of using $f_1 - 1$, but not $f_1 + 1$, the same property holds. In the cases where $f_1 - 1$ and $f_1 + 1$ are not used, since f_1 is the best possible assignment, it must be used. Finally, the simultaneous assignment of both $f_1 + 1$ and $f_1 - 1$, could lead to a better assignment than the ones using f_1 and other frequency f_2. For this reason, in order to ensure that the best assignment is achieved, individuals which use f_1, and individuals which use simultaneously $f_1 - 1$ and $f_1 + 1$ should be analysed. The way in which neighbours are generated ensure that both possibilities are explored, so the best assignment is achieved under such conditions.

Memetic Schemes

A set of metaheuristics that included the usage of the previous local search were designed. This research was done in collaboration with researchers of three universities: University of Malaga, University Carlos III of Madrid, and University of Extremadura. The following metaheuristics were tested:

- Steady-State Genetic Algorithm (SSGA)
- Scatter Search (SS)
- Local Search with Heuristic Restart (LSHR)
- Evolutionary Algorithm with Increasing Population Size (EAIPS)

From the previous metaheuristics, my research group was in charge of designing EAIPS. Thus, the internal operation of such an algorithm is presented. The basic operation of the rest of the algorithms has already been presented in Chapter 2. The details can be found in [166].
EAIPS is a memetic algorithm that combines a modified EA with a $(1+1)$ selection operator and the previously described local search. The algorithm has the ability to perform as a trajectory-based algorithm when no stagnation is detected. However, it increases the population size in order to avoid strong local optima when necessary, behaving then as a population-based algorithm.

Algorithm 19 Evolutionary Algorithm with Increasing Population Size

```
1: initialise(P)
2: P ← localSearch(P)
3: while not time-limit do
4:    offspring ← variation(P)
5:    offspring ← localSearch(offspring)
6:    for i = 0 to populationsize do
7:       if P(i) is blocked SoftBloq generations then
8:          P(i) ← offspring(i)
9:       else
10:         P(i) ← best(P(i), offSpring(i))
11:      end if
12:   end for
13:   if P is blocked HardBloq generations then
14:      if P.size < MaxPopSize then
15:         increase population size
16:      end if
17:   end if
18: end while
```

Algorithm 19 shows the pseudocode of the approach. Individuals are encoded as an array of integer values, p, where $p(x)$ is the slot assigned to the transceiver t_x. $InitPSize$ initial individuals are generated in a completely random way (line 1). For each gene, a random value among the admissible ones is assigned. On each generation, the approach applies a variation operator over the population (line 4). The variation step lies in the application of a mutation operator to each individual in order to produce new offsprings. The $(1 + 1)$ selection operator is deterministic and selects the best individual between an offspring and its parent. In order to improve on the behaviour of the approach when dealing with local optima, two improvements were considered. First, if after $SoftBloq$ generations the fitness of the current individual has not been improved on, the selection operator used during the generation is a $(1, 1)$, i.e. the offspring is selected independently of its fitness value (lines 6-12). Moreover, if after $HardBloq$ generations the fitness value of none of the individuals has been improved on, an extra new individual is introduced in the population (lines 13-17). During the following generations, each individual included in the population is evolved applying the aforementioned rules. In order to avoid an uncontrolled growth of the population, the maximum size of the population is limited to $MaxPopSize$.

Three different mutation operators were implemented and compared. They include both directed and random operators. The tested mutation operators were the following:

- UM: each gene - or transceiver assignment - is mutated with a probability p_m. In order to perform the new assignment to the gene, a random value among the admissible ones is selected.

- Mapping Mutation (MM): being F the set of accepted frequencies by any of the transceivers, a random bijection $m : F \leftrightarrow F$ is generated. Each transceiver assignment t_x is replaced with a probability p_m by the value $m(t_x)$, if $m(t_x)$ is an admissible value for the transceiver t_x.

- Neighbour-based Mutation (NM): first, a random TRX t_x is mutated. Then, its neighbours, i.e. the TRXs which interfere with t_x, or are interfered with by t_x, are mutated with a probability p_m. The previous steps are repeated N times, but the TRX is selected among those ones which are neighbours of the TRXs that have been mutated in the previous steps. Thus, the mutation operator focuses on one zone of the network.

Multiobjectivised approaches

The original FAP problem has been multiobjectivised with several schemes. Specifically the following multiobjectivisations have been tested: DCN, ADI, DBI, RANDOM, INVERSION and DBI-THR. The parameter th of the DBI-THR multiobjectivisation was set up to the value 0.9. In addition, the scheme was multiobjectivised considering problem-dependent information (*Dependent*). In the last case, the helper-objective is calculated in the following way. First, the original FAP objective function (f) is decomposed into two independent functions f_0 and f_1, such that $f = f_0 + f_1$. The decomposition is performed as follows. First, a table containing all possible interferences between each pair of TRXs is generated. Then, this table is ordered based on the cost of the appearance of each pair ρ. The resultant position of each ρ is denoted by i_ρ. The cost associated with each ρ is taken into account in the function f_{obj} where $obj = i_\rho \bmod 2$. Finally, f_0 is used as the helper-objective. Likewise, f_1 could have been used as the helper-objective.
In order to solve the multiobjectivised approaches, a multi-objective memetic algorithm has been used. Specifically, a memetic version of the NSGA-II has been applied. The memetic version of NSGA-II incorporates the previously defined local search after the variation stage. The same mutation operators than in the mono-objective approaches have been used. In the case of the crossover operators the

following ones have been tested: UM, and Interference-based Crossover (IX). The IX crossover operates as follows. First, a TRX t_x is randomly selected. Every gene associated to a TRX which interferes with t_x or is interfered with by t_x, including the gene which represents the own t_x, is inherited from the first parent. The remaining genes are inherited from the second parent. As usual, the inverse mapping is used to generate the second offspring.

Hyperheuristics

The hyperheuristics has also been used for dealing with the FAP. They have been used both with the mono-objective schemes (EAIPS), and with the multiobjectivised schemes. In both cases, the hyperheuristic MONO_WEIGHT has been used. In the case of EAIPS the different configurations has been made up by considering different variation stages. In the case of the multiobjectivised schemes, the different configurations have considered different ways of multiobjectivising the FAP and different variation schemes.

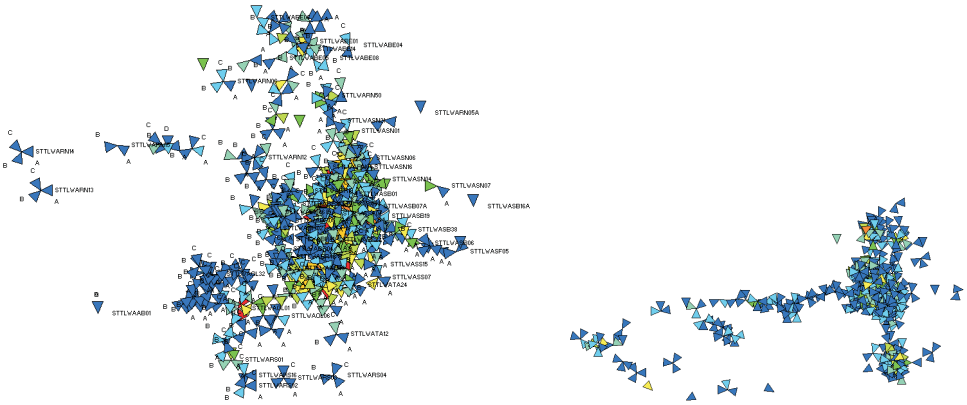
6.2.4 Experimental Evaluation

This section is devoted to present the experimental evaluation performed with the different proposals. The tests have been performed with real data that correspond to two US cities: Seattle and Denver. The former instance has 970 TRXs and 15 different frequencies to be assigned, while the latter has 2612 TRXs and 18 frequencies. The constants used in the mathematical formulation were set to $K = 100.000$, $c_{\text{SH}} = 6\ dB$, and $c_{\text{ACR}} = 18\ dB$. Figure 6.9 display the network topologies. The triangles represent a sectorised antenna in which several TRXs operate. The data source to build the interference matrix has used thousands of Mobile Measurement Reports (MMRs) rather than propagation prediction models.

Comparison of metaheuristics

This section shows the comparisons among the following algorithms: ACO, SSGA, SS, LSHR and EAIPS. The ACO approach was the one presented in [164]. The rest of the approaches have incorporated the usage of the specifically designed local search. First, some preliminary executions were performed with the aim of properly selecting the parameters of such approaches. The following parameterisation was used:

- SSGA: Population size = 10, uniform crossover with $p_c = 1.0$, random mutation with $p_m = 0.2$, selection with binary tournament, replacement = worst



(a) Seattle (b) Denver

Figure 6.9: Topology of the considered instances

individual

- SS: Population size = 40, RefSet size = 9, Solution combination method = uniform crossover
- EAIPS: initPSize = 1, softBloq = 50, hardBloq = 300, maxPopSize = 5, Neighbourhood-based Mutation with $p_m = 0.9$ and $N = 7$
- LSHR: Learning rate $f_r = 0.001$

Experiments have been carried out under exactly the same conditions: an Intel Xeon 3GHz processor and 2GB RAM have been used for this purpose. Since we are dealing with stochastic algorithms, we have carried out 30 independent runs for each metaheuristic. Moreover, the statistical tests have been performed with the short and long time ranges. Specifically, four different time limits (120, 600, 1800, and 3600 seconds) have been considered. Mean cost values, $\bar{x}$, and standard deviations, σ_n, of the 30 executions for every algorithm are summarised in Tables 6.4 and 6.5, for the Seattle and Denver instances, respectively (a grey background has been used to show the best value).
To properly put in context the quality of the generated plans, Table 6.6 displays the mean, $\bar{x}$, and the standard deviation, σ_n, of the cost of randomly generated solutions. In addition, the data for randomly generated solutions that undergo the local search method used by the metaheuristic algorithms is also shown. The goal is to highlight two different facts. First, the high accuracy of this local search algorithm, which is able to reduce the cost of random frequency plans several orders

Table 6.4: Results of the metaheuristics for 4 different time limits on the Seattle instance

Seattle	120 s	600 s	1800 s	3600 s
ACO	$1889.64_{\pm 118.97}$	$1578.65_{\pm 148.06}$	$1380.33_{\pm 128.33}$	$1330.49_{\pm 116.79}$
SSGA	$1894.67_{\pm 79.83}$	$1757.60_{\pm 87.49}$	$1676.12_{\pm 63.02}$	$1628.05_{\pm 51.17}$
SS	$2115.56_{\pm 191.43}$	$1606.85_{\pm 183.21}$	$1348.74_{\pm 145.95}$	$1238.68_{\pm 141.55}$
EAIPS	$1417.28_{\pm 141.94}$	$1142.65_{\pm 108.47}$	$989.30_{\pm 73.07}$	$917.43_{\pm 51.92}$
LSHR	$1677.73_{\pm 208.45}$	$1341.40_{\pm 123.48}$	$1194.13_{\pm 105.85}$	$1102.13_{\pm 115.49}$

Table 6.5: Results of the metaheuristics for 4 different time limits on the Denver instance

Denver	120 s	600 s	1800 s	3600 s
ACO	$93439.46_{\pm 1341.54}$	$92325.42_{\pm 1111.56}$	$90649.93_{\pm 740.02}$	$89875.66_{\pm 699.49}$
ssGA	$89540.41_{\pm 1008.14}$	$87850.79_{\pm 583.45}$	$86908.94_{\pm 386.37}$	$86870.40_{\pm 320.02}$
SS	$89401.36_{\pm 1091.63}$	$87233.42_{\pm 874.73}$	$86122.66_{\pm 666.58}$	$85525.17_{\pm 494.54}$
EAIPS	$89798.47_{\pm 1305.70}$	$87859.68_{\pm 1038.68}$	$86835.99_{\pm 1016.04}$	$86363.98_{\pm 790.68}$
LSHR	$92946.57_{\pm 1519.37}$	$89680.33_{\pm 902.81}$	$88646.53_{\pm 526.92}$	$88367.90_{\pm 406.18}$

Table 6.6: Mean and standard deviation of 30 random solutions and 30 executions of local search for the Denver and Seattle networks.

	Random	Local Search
	$\bar{x}_{\pm\sigma_n}$	$\bar{x}_{\pm\sigma_n}$
Seattle	$55.204,11_{\pm 1815.99}$	$3.692,66_{\pm 405,19}$
Denver	$115.577.436,48_{\pm 3.902.668,53}$	$105.155,60_{\pm 2.077,20}$

of magnitude (specially in the Denver instance, the larger one). Second, to show that the metaheuristics can profit from the hybridisation with this local search by reaching frequency plans provoking weaker interference than those obtained by the standalone local search. Indeed, if we consider the results included in Tables 6.4 and 6.5, it can be easily seen that the FAP costs of the plans are lower for all the algorithms, time limits, and instances undertaken.

Taking into account these two tables (Tables 6.4 and 6.5), it is remarkable that all the tested algorithms are able to keep improving on the solution quality in the two problem instances even after one hour of execution. These numerical values

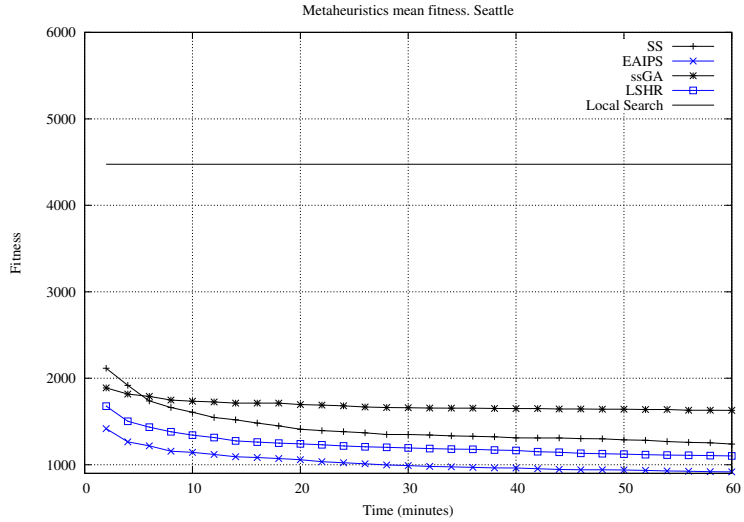


Figure 6.10: Mean interference of the four metaheuristics every 120 seconds in the Seattle network.

also show that the behaviour of the different algorithms depends on the real world instance addressed, even though the problem class is the same. Results reveal that in the Seattle instance, EAIPS is the best scheme. In this instance, the population size at the end of the executions has been between 1 and 3. Therefore, EAIPS has practically behaved as a trajectory-based approach. The second best-behaved scheme is LSHR, which is also a trajectory-based scheme.

For the Denver instance, the best plans are given by a population-based metaheuristic (SS), but the results of EAIPS are also competitive. In this case, the population sizes at the end of the executions of EAIPS are larger. Thus, at the end of the executions, EAIPS is behaving as a population-based approach. The strictly trajectory-based approach LSHR obtains the worst solutions.

The evolution of the mean of the costs obtained every 120 seconds is shown if figures 6.10 and 6.11. The good performance of EAIPS is confirmed with such figures. It can be appreciated that in both cases, competitive results are obtained.

To further analyse the results, a set of statistical tests were performed. Tables 6.7 and 6.8 show the results of the comparisons. The way to interpret the pairwise comparisons is as follows. For every pair of algorithms, the tables display those time ranges where differences are not significant. The "–" symbol means that all the differences are statistically significant. First, it must be noticed that for both problem instances, differences for the 3600 second time range are always significant. Second, it was noticed before that in the Seattle instance, trajectory-based

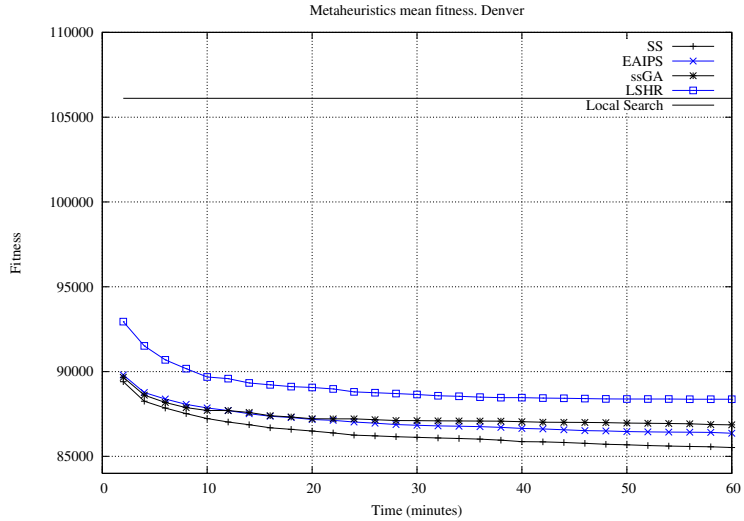


Figure 6.11: Mean interference of the four metaheuristics every 120 seconds in the Denver network.

algorithms performed better than population-based ones. Table 6.7 shows that, in fact, all differences between population-based and trajectory-based metaheuristics are significant. Moreover, differences among the population-based algorithms (ACO, SS, and ssGA) are in some cases not significant. With respect to the trajectory-based ones, EAIPS outperforms significantly LSHR for all time ranges, so this is clearly the best algorithm in this instance.

In the Denver instance, SS is the best algorithm for the 3600 second range, while it behaves similarly to other algorithms for shorter time spans. It seems that the diversity maintenance techniques of SS allow this metaheuristic to avoid stagnation for the longer time periods. It can also be seen that SSGA and EAIPS have a similar behaviour for most time ranges.

Summarising all the above, we can conclude that EAIPS and SS are particularly accurate algorithms for this domain. EAIPS is a very effective algorithm for both problem instances. It is significantly the best for Seattle and the runner up in Denver, and in this latter case, differences with the best performer (SS) are not statistically significant for time ranges smaller than 600 seconds. SS is the statistically significant winner for time ranges over 600 seconds in the Denver domain, and the third performer in Seattle. The remaining algorithms behave well in one of the domains (LSHR in Seattle and ssGA in Denver) but badly in the other.

Table 6.7: Post-hoc tests of the results for the Seattle instance. Time limits for which the pairwise comparison is not significant.

SSGA	120			
SS	600, 1800	–		
EAIPS	–	–	–	
LSHR	–	–	–	–
	ACO	ssGA	SS	EAIPS

Table 6.8: Post-hoc tests of the results for the Denver instance. Time limits for which the pairwise comparison is not significant.

SSGA	–			
SS	–	120, 600		
EAIPS	–	120, 600, 1800	120, 600	
LSHR	120	–	–	–
	ACO	SSGA	SS	EAIPS

Performance of multiobjectivisation

This section is devoted to perform a comparison between the results obtained with EAIPS, and the ones obtained with NSGA-II and multiobjectivisation. A set of 21 multiobjectivised approaches has been tested. They have been made up by combining seven multiobjectivisation schemes with three different variation schemes. The multiobjectivisation schemes have been: DCN, ADI, DBI, RANDOM, INVERSION, DBI-THR and DEPENDENT. In the first variation scheme, the UX operator with $p_c = 1$ has been used. In the second one, the IX operator with $p_c = 1$ has been applied. Finally, for the last variation scheme, the crossover operator has been disabled, i.e p_c has been fixed to 0. The population size was set up to 10 in every case. In order to identify the different schemes, the following nomenclature has been used: *Multiobjectivisation Scheme-Crossover operator*. In the cases where the crossover has not been used, the scheme is referred to using solely the name of the multiobjectivisation. Every variation schemes have used the next parameterisation for the Neighbour-based Mutation operator: $p_m = 0.9$ and $N = 7$. Each configuration has been executed during 4 hours.

Executions have been run on a Debian GNU/Linux computer with four AMD ® Opteron ™ (model number 6164 HE) at 1.7 GHz and 64 GB RAM. The compiler that has been used is GCC 4.4.5.

Figure 6.12 shows, for the Seattle and Denver instances, the evolution of the cost function mean values for the different multiobjectivised schemes. The results ob-

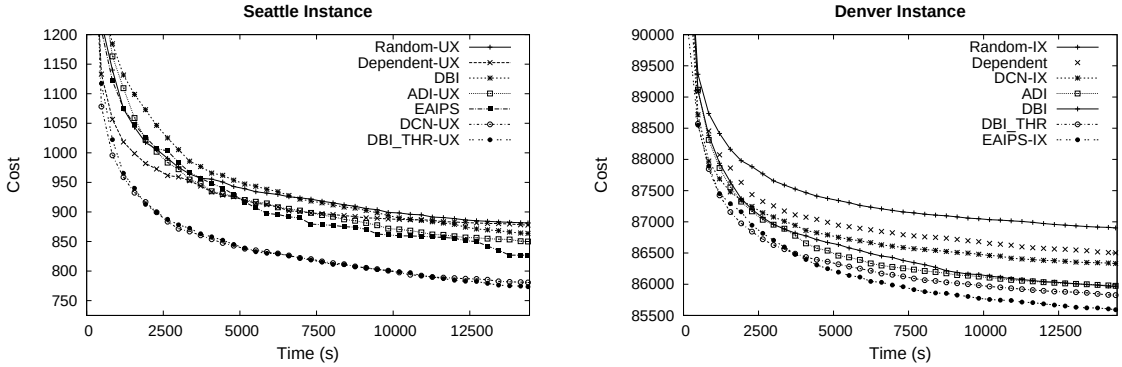


Figure 6.12: Mean cost obtained with different models for the Seattle and Denver instance

Table 6.9: Statistical Comparison of Configurations for the Seattle Instance

	DBI_THR-UX	DCN-UX	EAIPS	ADI-UX	DBI	Dep.-UX	Rand-UX
DBI_THR-UX	↔	↔	↑	↑	↑	↑	↑
DCN-UX	↔	↔	↑	↑	↑	↑	↑
VarPopEA	↓	↓	↔	↑	↑	↔	↑
ADI-UX	↓	↓	↓	↔	↔	↔	↔
DBI	↓	↓	↓	↔	↔	↔	↔
Dependent-UX	↓	↓	↔	↔	↔	↔	↔
Random-UX	↓	↓	↓	↔	↔	↔	↔

tained with EAIPS are also shown. Results are shown with the best behaved variation scheme for each approach. Since the *Reverse* multiobjectivisation has obtained very low quality results, it has not been taken into account. Table 6.9 shows whether the row configuration is statistically better (↑), not different (↔), or worse(↓), than the corresponding column configuration, in 4 hours of execution for the Seattle instance. Table 6.10 shows the same information for the Denver instance. For the Seattle instance, two multiobjectivised schemes have obtained better mean cost values than EAIPS. Moreover, in both cases the differences are statistically significant, showing the benefits of multiobjectivised techniques. For the Denver instance, the best mean results have been obtained by EAIPS-IX. However, differences between it and the best three multiobjectivised approaches are not statistically significant.

Taking into account the best multiobjectivised approach, Table 6.11 shows, for the Seattle instance, a statistical comparison among the three tested variation schemes. The same information is shown in Table 6.12 for the Denver instance. For the Seattle instance, no significant differences have been detected among different variation schemes. In the case of the Denver instance, the IX operator is statistically better

Table 6.10: Statistical Comparison of Configurations for the Denver Instance

	VarPopEA-IX	DBI_TH1	DBI	ADI	DCN-IX	Dep.	Rand-IX
VarPopEA-IX	↔	↔	↔	↔	↑	↑	↑
DBI_TH1	↔	↔	↔	↔	↑	↑	↑
DBI	↔	↔	↔	↔	↔	↑	↑
ADI	↔	↔	↔	↔	↑	↑	↑
DCN-IX	↓	↓	↔	↓	↔	↔	↑
Dependent	↓	↓	↓	↓	↔	↔	↔
Random-IX	↓	↓	↓	↓	↓	↔	↔

Table 6.11: Statistical Comparison of Variation Schemes for Seattle

	UX	No Crossover	IX
UX	↔	↔	↔
No Crossover	↔	↔	↔
IX	↔	↔	↔

Table 6.12: Statistical Comparison of Variation Schemes for Denver

	No Crossover	IX	UX
No Crossover	↔	↔	↑
IX	↔	↔	↑
UX	↓	↓	↔

than the UX operator, but not statistically different from the variation scheme with no crossover operator. Moreover, taking into consideration mean cost values, results of four schemes have been improved on by using the IX operator. Therefore, a deeper analysis with other instances should be performed to better explore the IX benefits.

In order to check the behaviour of the multiobjectivised approaches in the long term, a second experiment has been carried out. Specifically, the best multiobjectivised scheme and EAIPS, with its best variation scheme, have been executed using a stopping criterion of 24 hours. Figure 6.13 shows the boxplot of the cost values achieved in 4 and 24 hours for the Seattle and Denver instances. In the Seattle instance, for both values of the stopping criterion, the multiobjectivised approach is statistically better than EAIPS. In the Denver instance, considering 4 hours for the stopping criterion, both models are similar. In fact, differences are not statistically significant. In the case of 24 hours, most of the EAIPS-IX executions are better than the DBI_THR executions. However, some DBI_THR executions have been able to deal better with local optima. It can be appreciated that the variation in the results of DBI_THR is larger than the one obtained with EAIPS-IX. In fact, DBI_THR has obtained the best and the worst frequency plans.

Previous experiments have compared different mono-objective and multiobjectivised

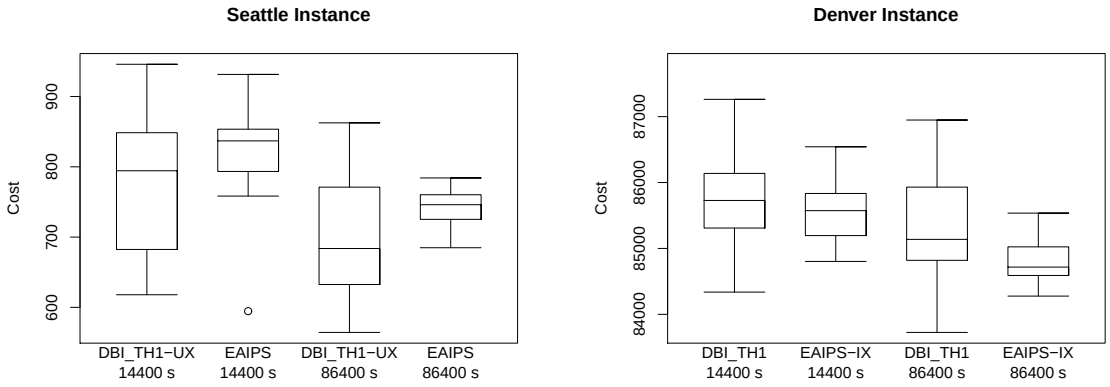


Figure 6.13: Boxplots of the obtained cost

configurations in terms of the quality achieved at fixed times. In addition, the run-time behaviour has been analysed by using the RLDs. In order to establish a high enough quality level, it has been fixed as the mean cost obtained in 8 hours of execution of EAIPS with its best variation scheme for each instance. Figure 6.14 shows the run-length distribution for both instances. In the Seattle instance, the 50% of executions of DBI_THR-UX has achieved the fixed quality level in 7440 seconds. In the case of EAIPS, the required time is 30840 seconds. Thus, the speedup factor to obtain such a success ratio is 4.14. By contrast, the success ratio of EAIPS is higher than the success ratio of DBI_THR-UX, considering 24 hours of execution. In the Denver instance, considering the 50% of the executions, EAIPS-IX is 1.97 times faster than DBI_THR. Moreover, considering the results for 24 hours, EAIPS-IX success ratio is higher than the DBI_THR success ratio. However, DBI_THR has been able to obtain higher success ratios than EAIPS-IX, when times lower than 5.5 hours are considered. Therefore, for the considered quality level, the most promising model depends on time constraints.

Hyperheuristic for mono-objective schemes

In the previous sections the proper behaviour of EAIPS has been demonstrated. However, in order to obtain such results, experiments with several parameterisations had to be performed. Thus, the computational and user effort required to obtain such results was very large. In this section the performance of the MONO_WEIGHT hyperheuristic when it has been used with the EAIPS scheme is analysed. Experiments with four and eight processors have been carried out. The same computational environment than in the previous experiment has been used.

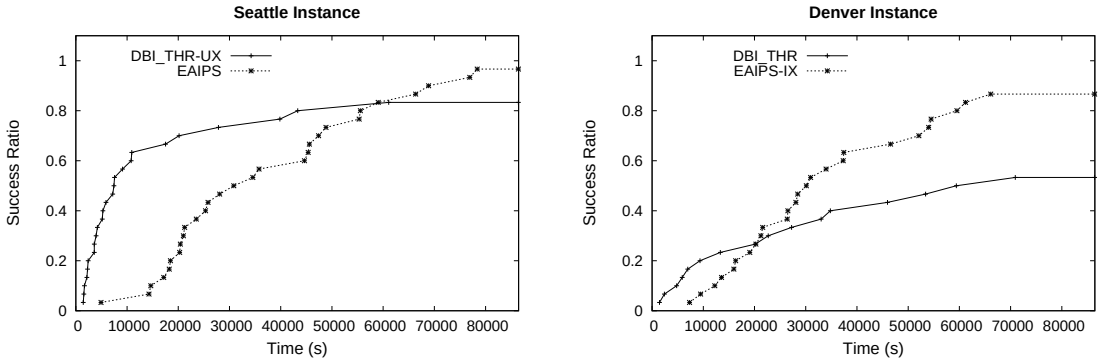


Figure 6.14: Run-length distribution for both considered instances

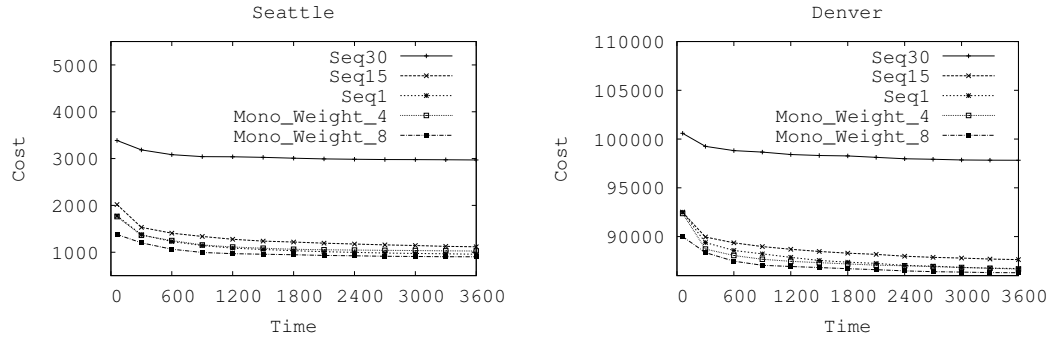


Figure 6.15: Evolution of the cost function for Seattle and Denver networks

First experiment performs a comparison of the costs of the frequency plans obtained by a set of sequential configurations of EAIPS and by the parallel approach of MONO_WEIGHT. The configuration of EAIPS was as follows: $InitPSize = 2$, $SoftBloq = 50$, $HardBloq = 300$, $MaxPopSize = 5$. Many configurations can be made up by using the set of defined mutation operators and by tuning their internal parameters. A set of 30 sequential configurations were executed and analysed. The set of mutation operator configurations was the following:

- UX with $p_m = \{0.1, 0.3, 0.5, 0.7, 0.9\}$
- MM with $p_m = \{0.1, 0.3, 0.5, 0.7, 0.9\}$
- NM with $(p_m, N) = \{(0.1, 1), (0.3, 1), (0.5, 1), (0.7, 1), (0.9, 1), (0.1, 3), (0.3, 3), (0.5, 3), (0.7, 3), (0.9, 3), (0.1, 5), (0.3, 5), (0.5, 5), (0.7, 5), (0.9, 5), (0.1, 7), (0.3, 7), (0.5, 7), (0.7, 7), (0.9, 7)\}$

Configuration	Seattle Index	Denver Index
NM (0.7, 7)	1	2
NM (0.9, 5)	2	1
NM (0.3, 3)	3	12
NM (0.5, 7)	4	5
NM (0.7, 5)	5	3
UM (0.1)	6	19
NM (0.5, 5)	7	17

Table 6.13: Robustness of sequential configurations

The parallel MONO_WEIGHT has been executed using the 30 described configurations as low-level meta-heuristics. Since the low-level approaches are not elitist, non-suitable configurations might degrade the quality of the population. For this reason, starting with the last population of the previous approach might not be adequate. If the new selected configuration is the same as the island current configuration, the execution continues with the last found population. Otherwise, the island configuration is updated and the changes performed by the algorithm over its subpopulation must be validated. In such a step, the model checks whether the individuals in the subpopulation has worsen its frequency plan cost more than 5% along the last configuration run. If an individual does not verify the condition, the original individual is recovered. This step is necessary because unsuitable configurations could excessively degrade the population quality.
MONO_WEIGHT was executed with the following parameterisation: $\beta = \frac{0.2}{30}$ and $k = 5$. The executions considered 4 and 8 worker islands. They are referred to as MONO_WEIGHT_4 and MONO_WEIGHT_8. Every sequential and parallel execution was run with a stopping criterion of 1 hour. For the parallel executions the local stopping criterion was fixed to 1 minute. Migration was performed following an asynchronous scheme with a migration probability of 1. The used migration topology was the ALL. Migrated individuals are selected following an elitist scheme, i.e. the best individual is selected to migrate. Replacements were performed also following an elitist scheme. They only take place when the migrated individual is better than any of the individuals in the new island. In such a case, the individual that is in the first position of the population is replaced.
Considering the obtained results, sequential algorithms were ordered based on the mean cost achieved at the end of the executions. An index based on such an order is assigned to each configuration. Therefore, for each instance, the best sequential execution will be referred to as "seq1", while the worst one will be referred to as

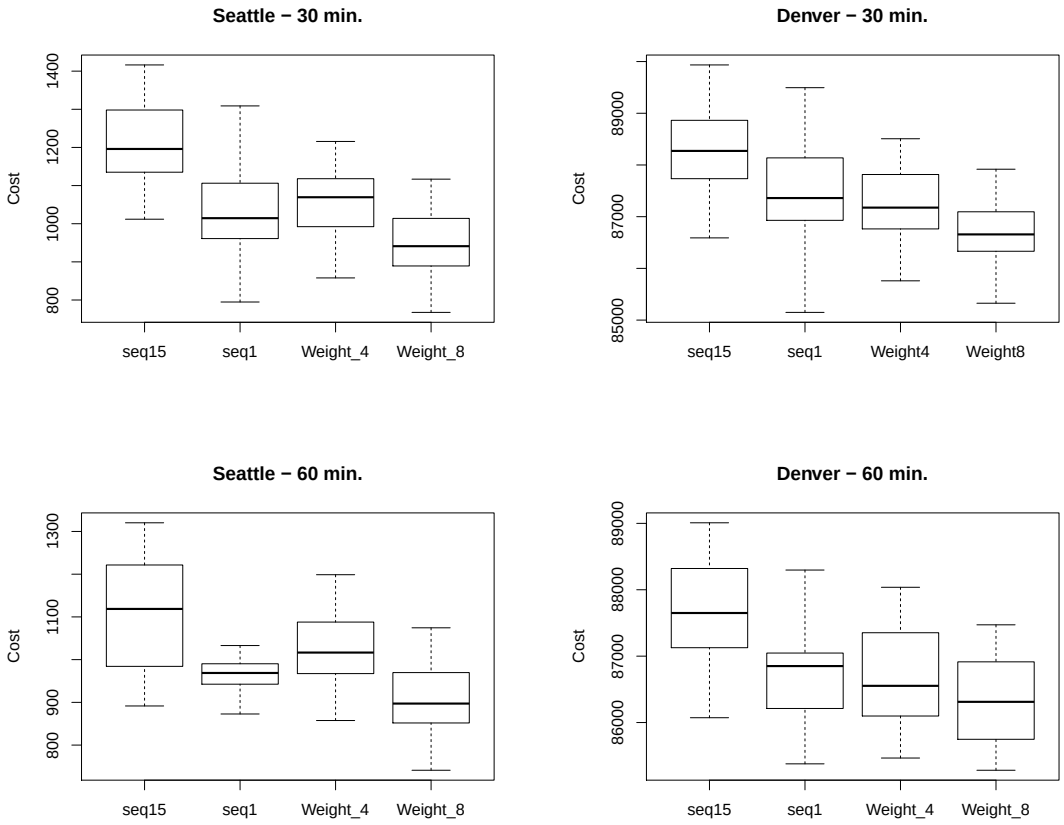


Figure 6.16: Box-plots of the achieved costs

"seq30". Table 6.13 shows the best configurations for the Seattle instance, and its corresponding index for the Denver instance. Most of the best configurations are suitable for both instances. However, some of them are not adequate, so they would produce a waste of resources if applied to the other instance. Some of the best configurations correspond to high values of p_m, while other ones correspond to low values. Thus, it is very difficult to know, a-priori, which configurations are suitable for a given instance.

Figure 6.15 displays, for both instances, the evolution of the mean cost achieved by MONO_WEIGHT, "seq1", "seq15", and "seq30". In both instances the costs achieved by MONO_WEIGHT$_4$ are very similar to the one achieved by the best sequential approach. In the Seattle network, "seq1" is slightly better than MONO_WEIGHT$_4$, while in the Denver network, MONO_WEIGHT$_4$ obtains better results than "seq1". Therefore, the parallel approaches, even with so few processors, can be used with

	Seattle			Denver		
	↑	↔	↓	↑	↔	↓
MONO_WEIGHT$_4$	7	6	17	0	3	27
MONO_WEIGHT$_8$	0	0	30	0	0	30

Table 6.14: Statistical analysis fixing the execution time

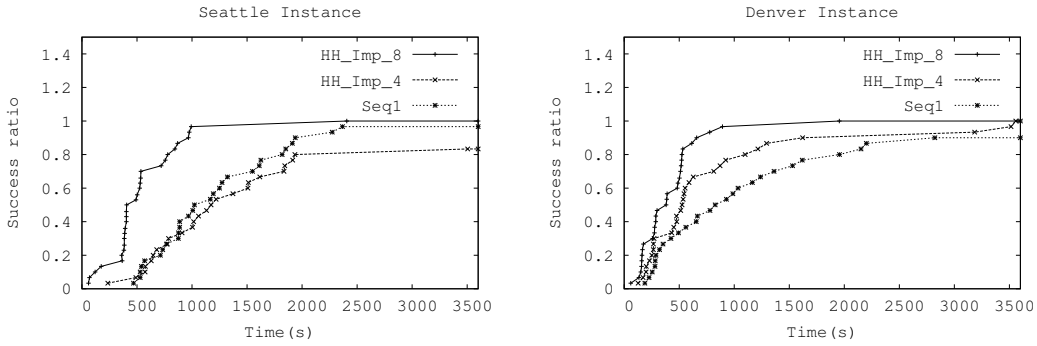


Figure 6.17: Run length distributions for "seq1" and MONO_WEIGHT

the aim of avoiding the testing of each one of the low-level configurations. Costs values achieved by "seq15" and "seq30" are clearly worse than the ones achieved by the parallel models. Executions with 8 worker islands produce much betters results. Differences between the models with 4 islands, and the ones with 8 islands are statistically significant in both instances. Therefore, they allow to avoid the testing of each one of the low-level configurations, and to speed up the achievement of high quality frequency plans. The boxplots of the costs obtained by the different models are shown in Figure 6.16. They were calculated using as stopping criterion 30 minutes and 1 hour. In both instances, the boxtplots confirm the similarity between "seq1" and the parallel models that use 4 worker islands. In addition, they show the superiority of the models with 8 worker islands.

It is also interesting to know the relative performance between the rest of the sequential configurations, and MONO_WEIGHT. Table 6.14 shows, for both instances, the number of sequential configurations which are better (↑), not different(↔), or worse(↓) than the corresponding row configuration. The comparison is performed in terms of the achieved fitness when considering a stopping criterion of 1 hour. It shows the better adaptation of the hyperheuristic to the Denver instance, than to the Seattle instance. In both instances, the parallel approach with 4 worker islands is better than most of the sequential configurations. When 8 worker islands are

Config.	mono_weight$_4$ speedup	mono_weight$_8$ speedup	Success Ratio (%)
seq1	1	2.5	100%
seq5	1.3	2.8	100%
seq10	2.4	4.9	100%
seq15	3.9	9.2	100%
seq20	6.5	13.1	100%
seq25	-	-	0%

Table 6.15: Speedup of the parallel models in the Seattle network

Config.	mono_weight$_4$ speedup	mono_weight$_8$ speedup	Success Ratio (%)
seq1	1.5	2.1	90%
seq5	1.8	2.6	90%
seq10	3.3	5.4	90%
seq15	3.9	5.8	90%
seq20	11.4	26.8	60%
seq25	-	-	0%

Table 6.16: Speedup of the parallel models in the Denver network

incorporated, the parallel approaches perform better than any sequential approach. The previous experiment has compared the schemes, mainly focused in terms of the achieved quality. However, since the parallel executions use more computational resources than the sequential ones, the improvement achieved by the parallel model must be quantified. In order to measure the improvement of the parallel approach, the run-time behaviour of the different models has also been analysed. The sequential configurations, as well as the parallel models, were executed using as finalisation condition the achievement of a certain level of quality. The quality level was established as the mean cost obtained by mono_weight$_4$ in 30 minutes. Since some of the sequential configurations are not able to reach such a quality level, a second stopping criterion - the execution of a maximum time of 10 hours - was also considered. Thus, the success ratio is defined as the probability of achieving the required quality level, considering a limitation in the execution time of 10 hours. Figure 6.17 shows the RLDs of the parallel models and of the best behaved sequential configuration when applied to the Seattle and Denver networks. In the case of the Seattle

	RANDOM$_4$				MONO_WEIGHT$_4$			
	Best	Worst	Mean	Median	Best	Worst	Mean	Median
30 min.	951	1316	1138	1149	857	1314	1063	1069
60 min.	949.5	1278	1074	1067	857	1198	1024	1016

Table 6.17: Quality comparison of the hyperheuristic-based models with a random scheme for the Seattle network

instance it shows the similarity among the RLD of the hyperheuristic-based method using 4 worker islands, and the best sequential configuration. Thus, such a parallel approach and the best sequential configuration require similar times to converge to a plan with the considered quality. The parallel scheme with 8 worker islands obtains such a quality level in less time. In the Denver network the hyperheuristics with only 4 worker islands require less time than the best sequential configuration to achieve similar quality levels. Therefore, for both instances the MONO_WEIGHT produce many benefits. In fact, by using only 4 processors, the success ratio achieved by the parallel models is similar - or even better - than the ones achieved by the best sequential approach. This demonstrates the good behaviour of MONO_WEIGHT, specially considering the large set of low-level metaheuristics incorporated in the model.

For the remaining configurations a summary of the analysis is shown. Table 6.15 shows, for the Seattle network, the success ratio and the speedup of the parallel models versus a set of selected sequential configurations. The speedup has been calculated considering the time required to achieving a 50% of success ratio. The speedup is marked with a line in the cases in which the sequential configurations were not able to achieve a success ratio greater than 50%. Although linear speedup is not achieved when comparing with the best configuration, it must be taken into account that when solving a problem, the best configuration is not known a priori, so, the time saving is much greater than the speedup calculated versus the best configuration. In fact, the speedup highly increases when comparing to other configurations. Moreover, the incorporation of more resources produces a faster convergence to high-quality results. The speedup achieved by the models which use 8 worker islands is about the double of the models which use 4 worker islands. Thus, with the usage of the first resources the testing of each low-level configuration can be avoided; and with the incorporation of more resources, the speedup can be improved. Table 6.16 shows the same information for the Denver network. The behaviour is very similar to the one detected in the Seattle instance.

Finally, it is important to check the suitability of the selection scheme performed by MONO_WEIGHT. The proposed hyperheuristic was compared with a strategy that randomly changes the configurations executed on the islands. Such a strat-

	RANDOM$_4$				MONO_WEIGHT$_4$			
	Best	Worst	Mean	Median	Best	Worst	Mean	Median
30 min.	86849	89174	87790	87756	85760	88507	87198	87175
60 min.	86390	88620	87224	87201	85465	88037	86677	86553

Table 6.18: Quality comparison of the hyperheuristic-based models with a random scheme for the Denver network

egy has been denoted by RANDOM$_4$. The involved configurations, migration scheme and stopping criteria were identical to the ones used in the first experiment. Tables 6.17 and 6.18 show the best, worst, mean and median of the costs achieved by MONO_WEIGHT$_4$, and RANDOM$_4$ approaches when applied to Seattle and Denver instances, respectively. In every case, the mean and median costs achieved by the new proposed models are better than the ones achieved by making a random mapping. Moreover, the statistical comparison of RANDOM$_4$ with MONO_WEIGHT$_4$ shows the superiority of MONO_WEIGHT$_4$. As shown, the incorporated hyperheuristic strategies produce an important improvement when compared to random selection schemes.

Hyperheuristic for multiobjectivised schemes

This section is devoted to perform an analysis of MONO_WEIGHT when it is used in conjunction with the previously described multiobjectivised schemes. One of the main drawbacks of the previous approach is that several multiobjectivisations have to be tested. The study revealed that the performance of the multiobjectivised schemes depend on the instance to solve. Therefore, the usage of MONO_WEIGHT might facilitate the application of multiobjectivisation and might reduce the computational and user effort required to solve new network instances. Moreover, it enables the usage of parallel environments. Tests were run on the HECToR machine, the UK's National Supercomputing Service. The processors used were AMD 2.3 GHz 16-core processors. The compiler used was GCC 4.6.1. Due to restriction in the computational resources, the amount of repetitions performed for each run has been 24 instead of 30.

The aim of the first experiment has been to analyse the capability of the hyperheuristic MONO_WEIGHT to obtain high-quality results in a single run while using a low number of processors. To do this, the MONO_WEIGHT model was executed with four different migration stages. They were generated by combining two topologies with two replacement selectors. The tested topologies were the RING and the ALL. The considered replacement schemes were the ELI and the HAM. In every case, an *elitist migration scheme* is applied. Specifically, a subpopulation individual is mi-

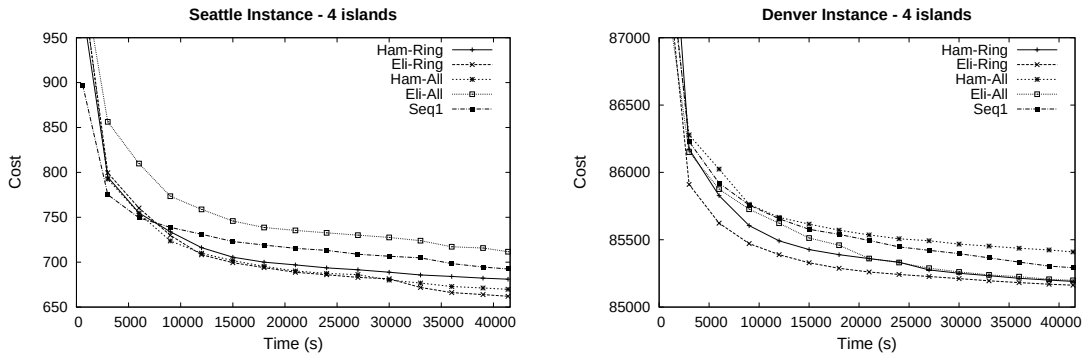


Figure 6.18: Evolution of the mean of the cost - 4 islands

grated when its cost is lower than the cost of any member of its previous generation. This is checked in every generation. The following nomenclature is used to identify the different migration stages: *Replacement_Scheme-Topology*. Tests in this experiment considered 4 worker islands ($n_p = 4$). The global stopping criterion was set to 11.5 hours of execution, while the local stopping criterion was set to 10 minutes. The MONO_WEIGHT model was applied with an adaptation level $k = 10$, and the value of β was set up such that 10% of the decisions performed by the hyperheuristic followed a uniform distribution, i.e. $\beta * n_h = 0.1$.

A set of 21 multiobjectivised configurations were used as low-level configurations. They were the same configurations than in previous experiments, i.e. they were made up by combining seven multiobjectivisation schemes with three different variation schemes. In order to compare the results of the MONO_WEIGHT model, the sequential versions of the aforementioned 21 low-level configurations were also executed. The stopping criterion was set to 11.5 hours. The low-level configurations were sorted based on the mean of the fitness achieved at the end of their executions. An index based on this order was assigned to each configuration. From now on, the best low-level configuration, i.e. the one that achieved the lowest mean cost, will be referred to as SEQ1, while the worst one will be referred to as SEQ21.

Figure 6.18 shows, for the Seattle and Denver instances, the trend in the mean of the cost obtained by the MONO_WEIGHT model with the four migration stages, and by the best low-level configuration. In both instances, three of the parallel models were able to improve on the results achieved by SEQ1. We see that although the MONO_WEIGHT model used more computational resources than the best sequential configuration, it achieved better results than SEQ1. By using the MONO_WEIGHT model, the requirement of testing each one of the low-level configurations under

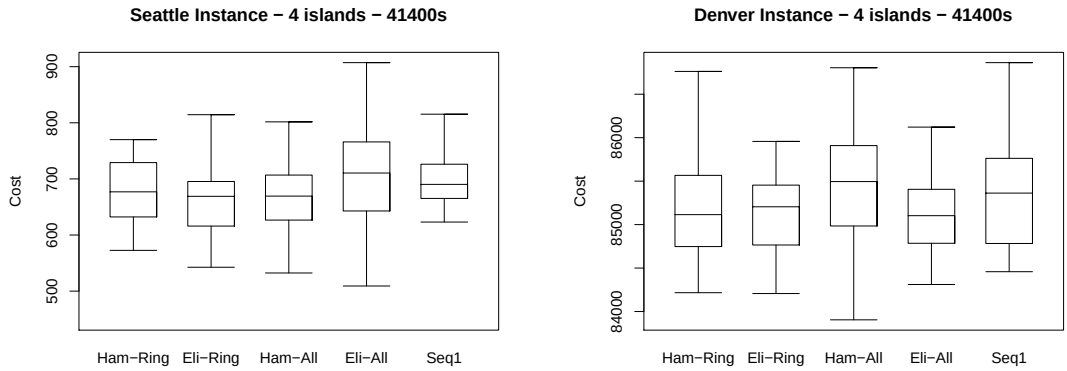


Figure 6.19: Boxplots for the Seattle and Denver Instances - 4 islands

Table 6.19: Statistical comparison for the Seattle Instance - 4 islands

	Ham-Ring	Eli-Ring	Ham-All	Eli-All	Seq1
Ham-Ring	↔	↔	↔	↔	↔
Eli-Ring	↔	↔	↔	↑	↔
Ham-All	↔	↔	↔	↔	↔
Eli-All	↔	↓	↔	↔	↔
Seq1	↔	↔	↔	↔	↔

consideration can be avoided. Thus, by using the MONO_WEIGHT model, the amount of computational resources saved can be quite considerable. In addition, since the parameter setting stage is alleviated, the user effort required to solve a new network instance is reduced.

In order to better analyse the results, the boxplots of the cost obtained by each model at the end of the runs are shown in Figure 6.19. For both instances, the results yielded by all the approaches were very similar. Table 6.19 shows, for the Seattle instance, whether the row model is statistically better (↑), not different (↔), or worse(↓) than the corresponding column model after 11.5 hours of execution. It reveals that the ELI-RING model is statistically better than the ELI-ALL model. There are no statistical differences among the remaining models. The same information is shown in Table 6.21 for the Denver instance. In this case no statistical differences appear. Consequently, regardless of the migration stage used, the MONO_WEIGHT model was able to yield competitive frequency plans.

A previous experiment compared different parallel models in terms of the quality achieved at fixed times. However, it is important to quantify the improvement achieved by these parallel approaches in terms of the amount of time saved. To do

Table 6.20: Speedup factors for the Seattle Instance - 4 islands

	Ham-Ring	Eli-Ring	Ham-All	Eli-All
Seq1	1.23	1.23	1.33	0.28
Seq3	1.23	1.23	1.33	0.28
Seq5	4.92	4.92	5.33	1.14

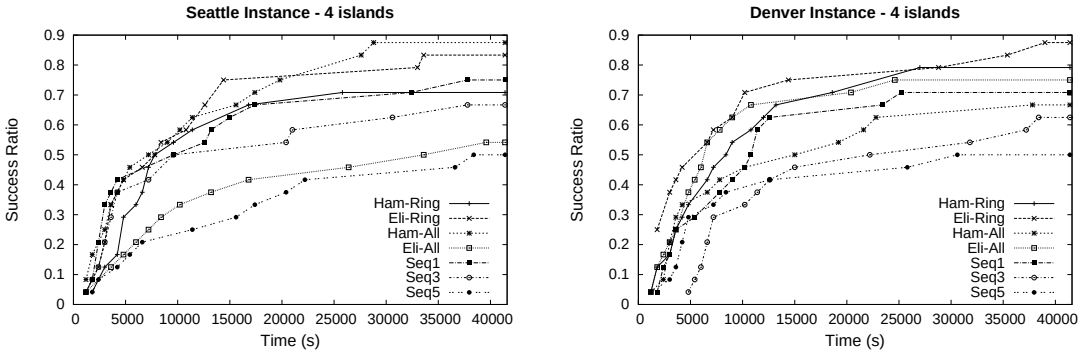


Figure 6.20: RLDs for the Seattle and Denver Instances - 4 islands

Table 6.21: Statistical comparison for the Denver Instance - 4 islands

	Ham-Ring	Eli-Ring	Ham-All	Eli-All	Seq1
Ham-Ring	↔	↔	↔	↔	↔
Eli-Ring	↔	↔	↔	↔	↔
Ham-All	↔	↔	↔	↔	↔
Eli-All	↔	↔	↔	↔	↔
Seq1	↔	↔	↔	↔	↔

Table 6.22: Speedup factors for the Denver Instance - 4 islands

	Ham-Ring	Eli-Ring	Ham-All	Eli-All
Seq1	1.28	1.63	0.72	1.63
Seq3	2.64	3.36	1.48	3.36
Seq5	3.64	4.63	2.04	4.63

so, the RLDs have been used. In order to establish a high enough quality level, it was set as the median of the cost obtained by SEQ5 in 11.5 hours. Figure 6.20 shows, for the Seattle and Denver instances, the RLDs of the parallel models, together with the RLDs of the SEQ1, SEQ3, and SEQ5 sequential configurations. It shows the similarities among the parallel models and SEQ1, validating the conclusions drawn

Table 6.23: Speedup factors for the Seattle Instance - 8, 16, and 32 islands

	8 Islands	16 Islands	32 Islands
Ham-Ring	6.37	12.75	17
Eli-Ring	5.66	10.2	12.75
Ham-All	0.75	5.66	7.28
Eli-All	1.02	4.25	2.68

Table 6.24: Statistical comparison for the Seattle Instance - 32 islands

	Ham-Ring	Eli-Ring	Ham-All	Eli-All	Seq1
Ham-Ring	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$	$\uparrow$	$\uparrow$
Eli-Ring	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$	$\uparrow$	$\uparrow$
Ham-All	$\downarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$
Eli-All	$\downarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
Seq1	$\downarrow$	$\downarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$

in the first experiment. However, some parallel models were able to achieve higher success ratios than SEQ1. In addition, the RLDs show the clear superiority of the parallel models compared to the remaining sequential configurations.
Considering the time required to attain a 50% success ratio, Table 6.20 shows, for the Seattle instance, the speedup factors obtained by the parallel models with respect to SEQ1, SEQ3, and SEQ5. The ELI-ALL parallel model was clearly outperformed by the other ones. In fact, its success ratio of 50% is four times slower than SEQ1 and SEQ3. The best parallel models obtained super-linear speedup when compared with SEQ5. This means that the hyperheuristic granted more computational resources to the best-behaved sequential approaches. Thus, the decision space was explored more efficiently. Table 6.22 shows the speedup factors for the Denver instance. Similar conclusions can be drawn for this instance. However, the HAM-ALL model was the worst-behaved in this case.
Finally, a set of experiments intended to analyse the scalability of the MONO_WEIGHT model was performed. In the previous experiments the MONO_WEIGHT model was executed with four islands. Differences among the results obtained with different migration stages are not very clear. However, for a larger number of islands, the differences might be more obvious. For instance, consider the RING migration topology in which each island sends its solutions to another island, versus the ALL migration topology, where each island sends its solutions to $n_p - 1$ islands. As n_p increases, so do the differences between the two topologies. This is why the scalability analysis was performed taking into consideration the four aforementioned migration stages. For this analysis, the MONO_WEIGHT model was executed with 8, 16, and 32 islands, and using the same parameterisation as in the previous experiments.
Considering the Seattle instance, the speedup factors for the different parallel mo-

Table 6.25: Speedup factors for the Denver Instance - 8, 16, and 32 islands

	8 Islands	16 Islands	32 Islands
Ham-Ring	11.5	13.8	17.25
Eli-Ring	11.5	17.25	23
Ham-All	8.62	13.8	5.75
Eli-All	6.9	7.66	9.85

Table 6.26: Statistical comparison for the Denver Instance - 32 islands

	Ham-Ring	Eli-Ring	Ham-All	Eli-All	Seq1
Ham-Ring	↔	↔	↑	↑	↑
Eli-Ring	↔	↔	↑	↑	↑
Ham-All	↓	↓	↔	↔	↑
Eli-All	↓	↓	↔	↔	↑
Seq1	↓	↓	↓	↓	↔

dels, using SEQ1 as the reference model, are shown in Table 6.23. In this case the quality level was set as the lowest median of the cost obtained by any of the parallel models. The speedup factors were calculated considering the time required to achieving a 50% success ratio. Note that differences among the models are very noticeable. For example, for 32 islands, the speedup values range from 2.68 to 17. Moreover, the parallel models which used the RING topology achieved the set quality level faster than the models that used the ALL topology. Specifying 32 islands and a stopping criterion of 11.5 hours, the statistical tests (Table 6.24) confirm the superiority of those MONO_WEIGHT models that apply the RING topology. The same analysis was carried out for the Denver instance. The speedup factors are shown in Table 6.25, while the results of the statistical tests are given in Table 6.26. In this instance, the superiority of the models that used the RING topology is also clear.
In this experiment, the quality of the frequency plans achieved by the parallel models, when a high number of islands are considered, was also analysed. Considering the Seattle instance, the boxplots of the best and worst parallel models, in terms of the speedup values obtained with 32 islands, are shown in Figure 6.21. In the case of the MONO_WEIGHT model with the HAM-RING migration stage, the incorporation of extra islands into the scheme produced a clear decrease in the interference cost. In contrast, when the ELI-ALL migration stage was incorporated, the MONO_WEIGHT model was unable to profit from the incorporation of more islands. In the case of the Denver instance, the boxplots for the best and worst parallel models are shown in Figures 6.22. For this network, the same conclusions can be drawn. However, the best and worst migration stages were the ELI-RING and HAM-ALL, respectively.
Finally, it is important to note that the best frequency plans for both instances, previously published in [164], have been improved upon with the models here con-

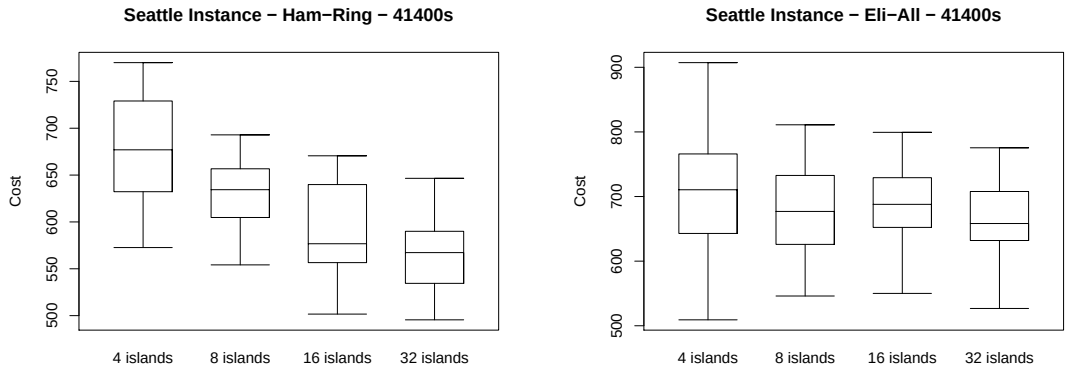


Figure 6.21: Boxplot for the Seattle Instance with the best and worst migration schemes

sidered. In the case of the Seattle instance the interference cost was decreased to 486.617. The best frequency plan obtained for the Denver instance has a cost of 83340.2. These frequency plans were obtained using the ELI-RING migration with 32 islands.

6.3 Broadcast Operation in Mobile Ad-hoc Networks

6.3.1 Introduction

MANETs [169] are fluctuating, self-configuring networks of mobile hosts, called *nodes* or *devices*, connected by wireless links. This kind of network has numerous applications because of its capacity of auto-configuration and its possibilities of working autonomously or connected to a larger network. No static network infrastructure is needed to support the communications between nodes, which are free to move arbitrarily. Devices in MANETs are usually laptops, PDAs, or mobile phones, equipped with network cards featuring wireless technologies. This implies that devices communicate within a limited range and also that they can move while communicating.
Broadcasting is a common operation at the application level and it is also widely used for solving many network layer problems. It is expected to be performed very frequently, serving also as a last resort to provide multicast services. Hence, having

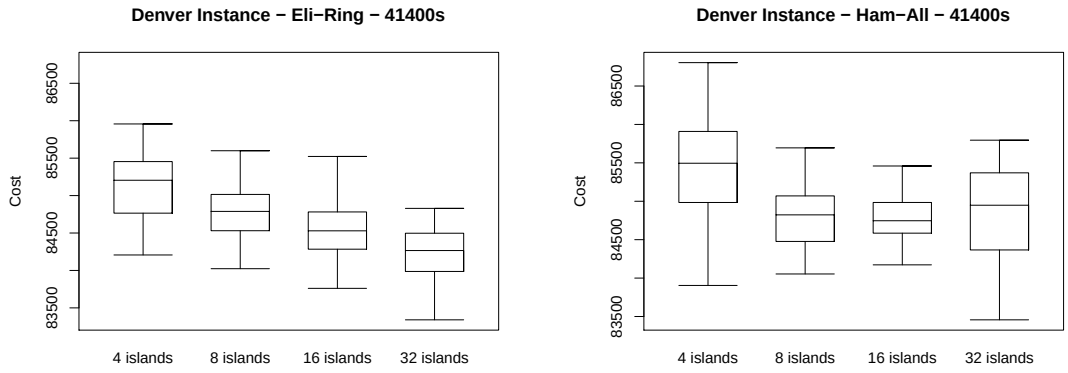


Figure 6.22: Boxplot for the Denver Instance with the best and worst migration schemes

a well-tuned broadcast strategy results in a major impact in network performance. The optimisation implies satisfying several objectives simultaneously: the number of reached devices (*coverage*) must be maximised, a minimum usage of the network (*bandwidth*) is desirable, and the process must take a time as short as possible (*duration*).

6.3.2 Problem Description

This research focuses on the study of the broadcast operation in a particular kind of MANETs, the *metropolitan* MANETs. These MANETs have some specific features that hinders the testing in real environments. First, the network density is heterogeneous. In addition, it is continuously changing because devices in a metropolitan area move and/or appear/disappear from the environment. For this reason, many simulation tools have been developed [124]. In this work the *Madhoc* simulator [123] has been used. This tool provides a simulation environment for several levels of services based on different types of MANETs technologies and for a wide range of MANET real environments. It also provides implementations of several broadcast algorithms [249]. From the existing broadcast protocols, the *Delayed Flooding with Cumulative Neighbourhood* (DFCN) [125] has been selected because it was specifically designed to deal with metropolitan MANETs.
DFCN is a deterministic and totally localised algorithm. It uses heuristics based on the information from one hop. Thus, it achieves a high scalability. The behaviour of each device when using DFCN is driven by three events: the reception of a

message (*reactive behaviour*), the expiration of the *random delay for rebroadcasting* (RAD) of a message, and the arrival of a new neighbour to its covered area (*proactive behaviour*). When one of these events occurs, DFCN reacts by behaving in a specific manner [125]. DFCN can be adapted to different environments by configuring a set of internal parameters. Although DFCN has shown good behaviour with metropolitan MANETs, the task of configuring such parameters is not trivial, and the proper operation of the protocol is sensitive to such a configuration. The set of parameters that must be configured is:

- *minG*: minimum gain for forwarding a message. The value must be in the range $[0.0, 1.0]$.
- [*lowerRAD*, *upperRAD*]: range values for the RAD. The values must be in the range $[0.0, 10.0]$.
- *proD*: maximum density for which it is still necessary to use proactive behaviour for complementing the reactive behaviour. The value must be in the range $[0, 100]$.
- *safeDensity*: maximum density below which DFCN always rebroadcasts. The value must be in the range $[0, 100]$.

Since the algorithm is used as a black-box, the internal operation of DFCN is not considered in the design of the optimisers. Given the values of the five DFCN parameters and a MANET scenario, the *Madhoc* tool does the corresponding simulation and provides an estimate of the three objectives: duration, coverage, and bandwidth. The goal is to find the parameters that optimise the DFCN behaviour, i.e. the set of non-dominated solutions. One possibility to find the most suitable configurations is to systematically vary each of the five DFCN parameters. However, the possible parameter combinations are too large and evaluations in the simulator are computationally expensive - each evaluations takes several seconds. Therefore, such a technique is unable to obtain good quality solutions in a reasonable amount of time. Other alternative lies in deeply analysing the problem to extract information to define a heuristic strategy, but the complexity and stochastic behaviour of the given problem hinders it. For these reasons, one usual way of affording this problem is through evolutionary techniques [9].

6.3.3 Proposed Optimisation Schemes

This problem was solved in collaboration with researchers of other universities. Specifically, the research was done in collaboration with researchers of University of

Malaga, University Carlos III of Madrid, and University of Extremadura. Each research group implemented and integrated several metaheuristics in the METCO tool. Then, they were executed collaboratively using the hyperheuristic HV_WEIGHT. The set of tested metaheuristics were the following:

- Strength Pareto Evolutionary Algorithm 2 (SPEA2).
- Non-Dominated Sorting Genetic Algorithm II (NSGA-II).
- Indicator-Based Evolutionary Algorithm (IBEA): The non-adaptive and adaptive versions were considered.
- Evolution Strategy with NSGA-II (ESN).
- Multi-Objective Particle Swarm Optimisation (MOPSO).
- Multi-Objective Cellular Genetic Algorithm (MOCell).
- Non-dominated Sorting Differential Evolution for Multiobjective Optimisation (NSDEMO).

The variation scheme of the previous algorithms - for the ones that do not define a specific variation scheme - was based on using a mutation and a crossover operator. They were the PM and the SBX operators. Regarding the encoding of the individuals, a direct encoding was used. In order to perform a correct simulation, the value of lowerRAD must be lower than the value of upperRad. In order to force it, the values of their corresponding genes are interchanged when required.

6.3.4 Experimental Evaluation

In this section the experimental evaluation performed for the broadcast optimisation problem is presented. The experimental analysis has been performed with an environment that represents a mall MANET scenario. The main aim of the analysis has been to validate the proper behaviour of a parallel scheme based on the HV_WEIGHT hyperheuristic. For doing it, such a hyperheuristic has been executed using the previously presented metaheuristics as low-level configurations. The HV_WEIGHT approach has been compared with the sequential version of the metaheuristics and with other PMOEAs. Specifically, for each implemented metaheuristic a homogeneous island-based scheme is considered. Each homogeneous scheme is referred to as "HOMO-ALGORITHM". Also, a heterogeneous scheme that considers the eight implemented metaheuristics has been executed. It is named "*heterogeneous*". Every parallel model was executed considering 8 islands. The subpopulation size on each

Table 6.27: Mean hypervolume achieved by the different PMOEAs

Parallel	**Evaluations limit**		
Model	5000	10000	25000
HV_WEIGHT	0.725	0.737	0.746
heterogeneous	**0.711**	**0.721**	**0.729**
homo-SPEA2	0.713	**0.724**	0.738
homo-NSGA-II	**0.712**	**0.723**	**0.739**
homo-IBEA	0.715	**0.724**	0.733
homo-adap-IBEA	0.716	0.725	0.733
homo-ESN	**0.707**	**0.718**	**0.726**
homo-MOPSO	**0.682**	**0.683**	**0.685**
homo-MOCell	**0.690**	**0.702**	**0.712**
homo-NSDEMO	**0.713**	**0.720**	**0.727**

island has been fixed to 15 individuals, while the population size for every sequential execution has been fixed to 100 individuals. The remaining parameterisation of each metaheuristic was as follows:

- SPEA2: $p_m = 0.2$, $p_c = 0.9$
- NSGA-II: $p_m = 0.2$, $p_c = 0.9$
- IBEA, adaptive-IBEA: $p_m = 0.2$, $p_c = 0.9$, k = 0.002
- ESN: $\sigma = 0.1$
- MOPSO: $p_m = 0.2$, divisions in archive = 30
- MOCell: $p_m = 0.2$, $p_c = 0.9$
- NSDEMO: $F = 0.5, CR = 1.0$

In every execution the same migration scheme was specified. The topology was the ALL. The migration probability has been fixed to 0.05 and the number of individuals to migrate was limited to 4 each time. The migration scheme was the ELI-RAND, and the replacement selector was the ELI100. The global stopping criterion for every execution was 25.000 evaluations. HV_WEIGHT was executed with the following parameterisation: $\beta = \frac{0.2}{8}$ and $k = 5$. The local stopping criterion was fixed to 15 generations. In all cases, the final solution was limited to 100 elements.
Tests have been run on a Debian GNU/Linux cluster of 8 Intel® Xeon™ 3.20 Ghz bi-processor nodes with 1Gb RAM. The interconnection network is a Gigabit Ethernet. The compiler and MPI implementation used were *gcc 3.3* and MPICH *1.2.7*.
The first experiment compares the different aforementioned PMOEAs among them. For each type of execution, 30 repetitions have been performed. In order to detect

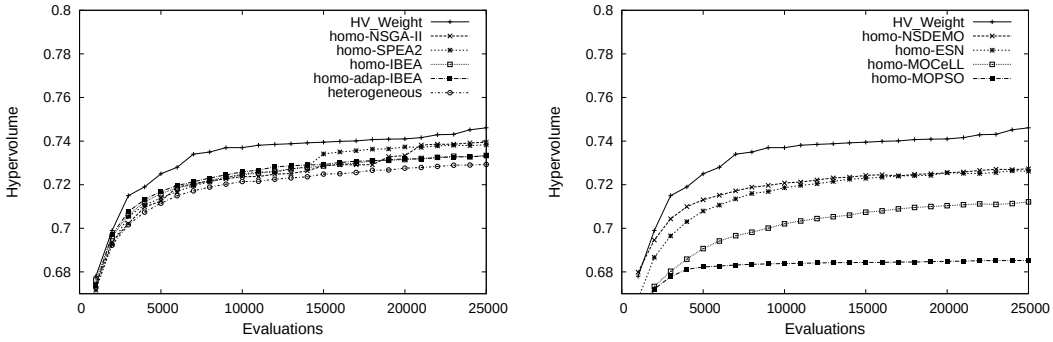


Figure 6.23: Hypervolume achieved by the parallel models

differences between the algorithms within short and long time ranges, three different number of individual evaluation limits have been considered: 5.000, 10.000 and 25.000 evaluations. The computing time of a sequential execution with 25.000 evaluations is approximately 50 hours. Table 6.27 shows the mean hypervolume attained by each parallel model at the given limits. The HV_WEIGHT configuration achieves the best results. The dynamic mapping allows giving more computational resources to the most suitable algorithms, thus improving upon the results of the heterogeneous model. Moreover, the simultaneous usage of different evolutionary algorithms makes it possible to combine the benefits of each one. Therefore, the results of every homogeneous island-based model are also improved on. Figure 6.23 presents, for each model, the mean hypervolume achieved during the executions. It can be observed the superiority of HV_WEIGHT for any considered amount of evaluations.
As our interest is mainly focused on the HV_WEIGHT model, it has been statistically compared with the remaining PMOEAs. Table 6.27 shows data in bold when differences between the row model and the HV_WEIGHT model are significant. The parallel hyperheuristic achieves a better hypervolume in every case. In the case of 10.000 evaluations, all differences are significant, except with *homo-adap-IBEA*, showing the good performance of the approach.
The second experiment analyses the run-time behaviour of the sequential and parallel models. Each sequential configuration, as well as the HV_WEIGHT and the heterogeneous models, were executed using as finalisation condition the achievement of a certain level of hypervolume quality: the median obtained by "heterogeneous" in 20.000 evaluations in the first experiment. In this way, a high hypervolume is used, but no so high so that other approaches are not able to reach it. A second stopping criterion - the execution of a maximum number of 25.000 evaluations - was also considered. Figure 6.24 shows the RLDs of the models that obtained a success ratio

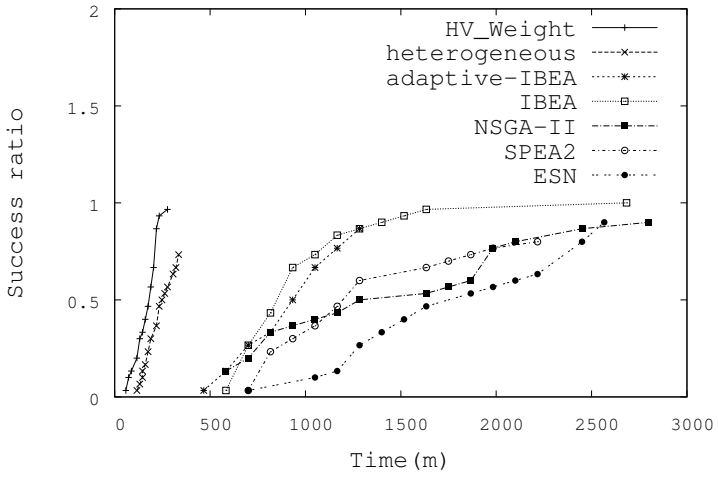


Figure 6.24: Run length distributions of parallel and sequential approaches

Table 6.28: Speedup of the proposed model and success ratio for sequential models

Sequential Models	**Heterogeneous Speedup**	**hv_weight Speedup**
adap-IBEA	3.76	4.92
IBEA	3.76	4.92
NSGA-II	5.17	6.76
SPEA2	5.17	6.76
ESN	7.52	9.84

higher than 50%. The advantages of using the parallel models are clear. Table 6.28 shows the speedup factors of the parallel approaches with respect to the different sequential schemes. The factors have been calculated considering the time required to obtaining a 50% of success ratio. The model based on HV_WEIGHT has obtained better speedup factor than the heterogeneous scheme. Moreover, high speedup factors have been obtained, demonstrating the adequate performance of HV_WEIGHT for the considered optimisation problem.

Chapter
7

Two-dimensional Packing Problem

This chapter is devoted to present a set of problem-dependent techniques for a two-dimensional packing problem. Such techniques have been merged with the problem-independent schemes previously presented. The specific version of the packing problem was proposed in the Genetic and Evolutionary Computation Conference. Results show the validity of the proposals.

7.1 Introduction

Bin Packing problems are combinatorial NP-hard problems in which objects with different shapes, volumes and/or areas must be packed into a finite number of bins. They are closely related to cutting problems. In cutting problem the main goal is to cut large stock sheets into a set of smaller pieces. Cutting and packing problems can be classified [86, 251] according to several characteristics. Some of the most used ones are the number of dimensions (1D, 2D, 3D), the number of available patterns, the shape of the patterns (regular or irregular), the orientation, and the objective that must be optimised. Some popular variants are: 2D strip packing [81], constrained 2D cutting stock [242], knapsack problems [172], packing with cost [46], and online packing [216]. Cutting and packing problems have many applications and are widely used inside more complex systems, e.g. filling up containers, optimisation of the layout of electrical circuits and multiprocessor scheduling.
In the last years, a competition session has been organised inside the Genetic and Evolutionary Computation Conference (GECCO). Since many participants try to explore the contest problems, this kind of competition is of great value. Researchers can apply new advances in the field, thus allowing to compare a large set of different proposals working on a given specific problem. The contest problems are usually

proposed in a way that different areas of evolutionary computation may be explored. In the GECCO 2008 competition session a variation of a 2D bin packing problem was proposed. In this research a set of schemes for dealing with such a problem has been designed. The current best-known solution for the instance proposed in the contest has been found using the proposals here described.

7.2 Problem Description

The Two-Dimensional Packing Problem (2DPP) is a two-dimensional variation of a bin packing problem. Problem instances are described by the following data:

- The sizes of a rectangular grid: X, Y.
- The maximum number which can be assigned to a grid position: N. The value assigned to each grid location is an integer in the range $[0, N]$.
- The scoring or value associated to the appearance of each pair (a, b) where $a, b \in [0, N]$: $v(a, b)$. Note that $v(a, b)$ need not be the same as $v(b, a)$.

A candidate solution is generated by assigning a number to each grid position. The objective of the proposed problem is to best pack a grid so that the sum of the point scores for every pair of adjacent numbers is maximised. Two positions are considered to be adjacent if they are neighbours in the same row, column, or diagonal of the grid. Once that a particular pair is collected, it cannot be collected a second time in the same grid.
Mathematically, the problem objective is to find the grid G which maximises the fitness function f:

$$f = \sum_{a=0}^{N} \sum_{b=0}^{N} v_2(a, b)$$

where

$$v_2(a, b) = \begin{cases} 0 & if\ (a, b)\ are\ not\ adjacent\ in\ G \\ v(a, b) & if\ (a, b)\ are\ adjacent\ in\ G \end{cases}$$

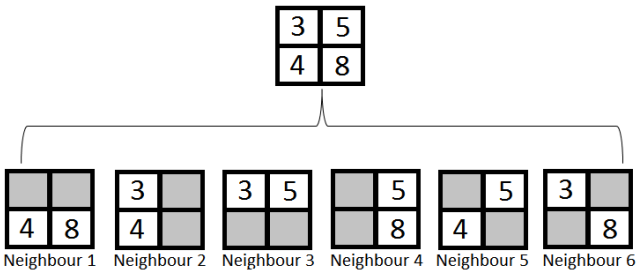


Figure 7.1: Generation of neighbours by the learning process

7.3 Proposed Optimisation Schemes

7.3.1 Local Search

A local search specifically tailored for the 2DPP has been designed. It is based on a mono-objective stochastic hill-climbing local search. The application of a local search allows admissible solutions to be achieved in relatively short times. The applied local search strategy has the following features. For each pair of adjacent grid positions (i, j) and (k, l), a neighbour is considered. This is illustrated in Figure 7.1. Each neighbour is generated by assigning the best possible values to the positions (i, j) and (k, l) (shading positions in Figure 7.1), leaving intact the assignments in any other grid location. In order to assign the best values to both locations, the trivial solution involves enumerating all possible pairs for a later selection of the best one. As such approach is computationally too expensive, a mechanism to prune the explored values has been used. First, all the possible assignments $n \in [0, M]$ to the grid position (i, j) are considered, and the contribution of each assignment $v_{ij}(n)$, assuming position (k, l) unassigned, is calculated. The same process is performed for the position (k, l), assuming the position (i, j) unassigned, and thus calculating $v_{kl}(n)$. The contribution to the objective function obtained by assigning a value a to the position (i, j), and a value b to the position (k, l), is given by:

$$v_{ij}(a) + v_{kl}(b) + v'(a, b) - v_{rep} \tag{7.1}$$

where $v'(a, b) = v(a, b) + v(b, a)$ if the pair (a, b) was not already in the grid, or 0 if it was, and v_{rep} is the value associated to pairs that are created by both, the assignment of the value a to (i, j) and the assignment of the value b to (k, l), which must be considered only once. An upper bound of such a contribution is given by:

$$v_{ij}(a) + v_{kl}(b) + min(bestV(a), bestV(b)) \tag{7.2}$$

where $bestV(n)$ is the maximum value associated to any pair (n, m), $m \in [0, M]$, i.e. $\max\{(v(n,m)+v(m,n)\}$. Being $bestObj$ the best objective value currently achieved for an assignment of the positions (i, j) and (k, l), the only values a', b' that must be considered are the ones that satisfy the following inequality:

$$v_{ij}(a') + v_{kl}(b') + min(bestV(a'), bestV(b')) > bestObj \tag{7.3}$$

By omitting the values in which the previous inequality is not satisfied, the assignments to be considered are highly reduced, so a large amount of time can be saved.
Since stochastic hill-climbing has been applied, the order in which neighbours are analysed is determined in a random way. The local search moves to the first new generated neighbour that improves on the current solution. Finally, the learning process stops when none of the neighbours improve on the current solution.

7.3.2 Memetic Schemes

The previous local search has been integrated in a memetic evolutionary approach. Specifically, the EAIPS scheme described in Chapter 6 has been used. The variation scheme uses a crossover and a mutation operator. The considered crossover operator was the Two-Dimensional Substring Crossover (SSX). The tested mutation operators were the following:

- *Uniform Mutation with Range (*UMR*)*: Each gene is mutated with a probability between min_p_m and max_p_m. In order to perform the new assignment to the gene, a random value among the admissible ones is selected.

- *Uniform Mutation with Domain Information (*UMD*)*: Each gene is mutated with a probability between min_p_m and max_p_m. In order to perform the new assignment to the gene, a random value is selected among the ones that produce a nonzero increase in the fitness value.

- *Selective Uniform Mutation (*SUM*)*: First, the UMR operator with probabilities min_p_m and max_p_m is applied. Then, the chromosome is considered as divided in rectangular windows of sizes between min_w and max_w. The fitness value associated to each considered window is calculated. The genes in the window with lowest fitness contribution are randomly mutated with a probability between $minw\_p_m$ and $maxw\_p_m$.

- *2D Rotation Mutation (*2DROT*)*: It is a generalisation of the rotation mutation suggested in [175]. First, it randomly decides to do column or row rotation.

Then, it generates a number n, between 0 and $X-1$, when column rotation is selected or between 0 and $Y-1$, when row rotation is selected. Finally, the columns or rows are rotated n positions, i.e. they are shifted, and the ones that are pushed off are reinserted at the other end of the grid. Since this operator does not introduce new values in the grid, it is always used in combination with any of the other mutation operators.

7.3.3 Multiobjectivised Approaches

A set of multiobjectivised approaches have also been tested. Specifically, the following multiobjectivisations have been considered: DCN, ADI, DBI, RANDOM, INVERSION, DCN-THR, and DBI-THR,. The parameter th of the DCN-THR and DBI-THR multiobjectivisations was set up to the value 0.99. In addition, a multiobjectivisation which considers problem-dependent information (Dependent) has also been tested. In such a case, the helper-objective has been calculated in the following way. First, the original 2DPP objective function (f) has been decomposed into two independent functions f_0 and f_1, so that $f = f_0 + f_1$. The decomposition is performed in the following way. First, a table containing all possible pairs whose score is not equal to zero is calculated. Then, this table is sorted based on the score of the appearance of each pair ρ. The resultant position of each ρ, after the sort, is denoted as i_ρ. The value associated to each ρ is taken into account to calculate the function f_{obj} where $obj = i_\rho \bmod 2$. Finally, f_0 is used as the helper-objective. Likewise, f_1 could have been used as the second objective.
In order to solve the multiobjectivised approaches, a multi-objective optimisation algorithm is required. Two memetic versions of MOEAs have been used. Specifically, the SPEA2 and NSGA-II have been considered. They have integrated the previously described local search after the variation scheme. In this case, the variation stage has been based on using the SSX crossover operator and the UMD mutation operator.

7.3.4 Hyperheuristics

The hyperheuristics has also been used for dealing with the 2DPP. They have been used both with the mono-objective schemes (EAIPS), and with the multiobjectivised schemes. In both cases, the hyperheuristic MONO_WEIGHT has been used. In the case of EAIPS the different configurations has been made up by considering different variation stages. In the case of the multiobjectivised schemes, the different configurations have considered different ways of multiobjectivising the 2DPP, and different MOEAs.

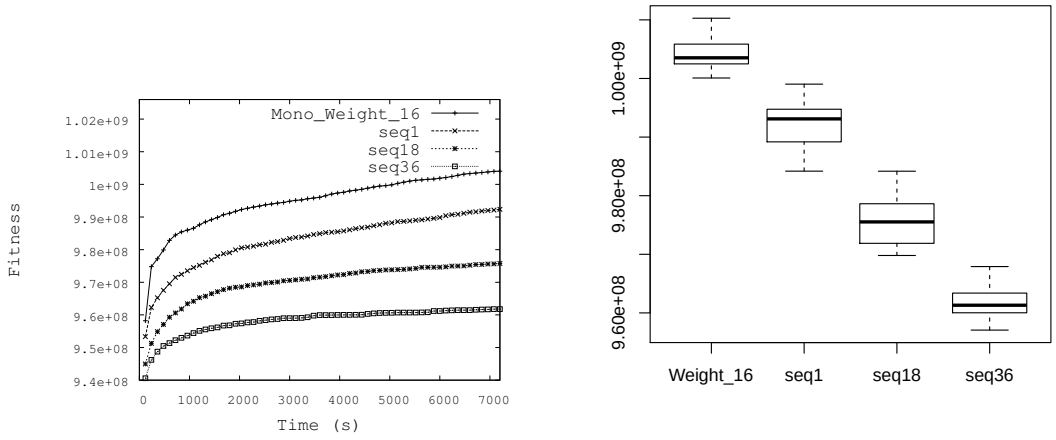
7.4 Experimental Evaluation

This section is devoted to present the experimental evaluation performed with the different proposals. Most of the analysis has been performed with the instance proposed for the contest. Such an instance has the following features: $X = 20$, $Y = 20$, $N = 399$, and 15962 possible pair scores. The search space of the contest instance consists of 400^{400} candidate solutions. The search space can be pruned, making use of bounds and other exact techniques. However, it is very difficult to propose an exact approach capable of solving such an instance in a reasonable amount of time.
In addition, another instance has been recently published. Some analyses with such an instance have also been performed. In the rest of the chapter we will name it "small instance". The small instance is characterised by the following parameters: $X = 10$, $Y = 10$, $N = 99$, and 9032 possible pair scores.
Tests were run on the HECToR machine, the UK's National Supercomputing Service. The processors used were AMD 2.3 GHz 16-core processors. The compiler used was GCC 4.6.1.

7.4.1 Mono-objective Schemes

This section is devoted to present the results obtained with the mono-objective schemes. The instance proposed for the contest has been used. First experiment performs a comparison of the fitness achieved by a set of sequential mono-objective configurations of EAIPS and by a parallel version of MONO_WEIGHT that uses such configurations as low-level schemes. The parameters of EAIPS have been set up as follows: $InitPSize = 2, SoftBloq = 50, HardBloq = 300, MaxPopSize = 10$. A set of 36 sequential configurations were executed and analysed. The set was made up by combining 18 different kinds of mutations with 2 different kinds of crossovers:

- 18 different mutations:
  - UMR with $(min\_p_m,\ max\_p_m) = \{(0.01, 0.1),\ (0.1, 0.15),\ (0.15, 0.2)\}$ together with 2DROT or alone.
  - UMD with $(min\_p_m,\ max\_p_m) = \{(0.01, 0.1),\ (0.1, 0.15),\ (0.15, 0.2)\}$ together with 2DROT or alone.
  - SUM with $(min\_p_m, max\_p_m, minw\_p_m, maxw\_p_m) = \{(0.01, 0.1, 0.1, 0.15),\ (0.1, 0.15, 0.15, 0.2),\ (0.15, 0.2, 0.2, 0.25)\}$ and $(min\_w,\ max\_w) = (3, 6)$ together with 2DRot or alone.
- 2 different crossovers:



(a) Evolution of the mean of the fitness (b) Boxplots in two hours of execution

Figure 7.2: Fitness obtained in the Contest instance

– No crossover

– SSX with $p_c = 1$.

In addition, a parallel version of MONO_WEIGHT that uses the 36 described configurations was run considering 16 processors. It will be referred to with the label MONO_WEIGHT$_{16}$. The sequential and parallel algorithms were executed with a stopping criterion of 2 hours. For the parallel algorithm the local stopping criterion was fixed to 2 minutes. An asynchronous migration scheme with a migration probability of 1 was defined. The ALL topology was used. Migrated individuals are selected following an elitist scheme, i.e. the best individual is selected to migrate. Replacements were performed following an elitist scheme. They only take place when the migrated individual is better than any of the individuals in the new island. In such a case, the individual which is in the first position of the population is replaced.
Sequential algorithms were ordered based on the mean fitness achieved at the end of the executions. An index based on such an order is assigned to each configuration. The best sequential execution will be referred to as "seq1", while the worst one will be referred to as "seq36". Figure 7.2 displays - at the left - the evolution of the mean fitness achieved with the parallel model and with "seq1", "seq18", and "seq36", i.e. the best, median and worst sequential configurations. The parallel execution performs much better than any of the sequential configurations. The boxplots obtained at the end of the executions are also shown in such a figure.

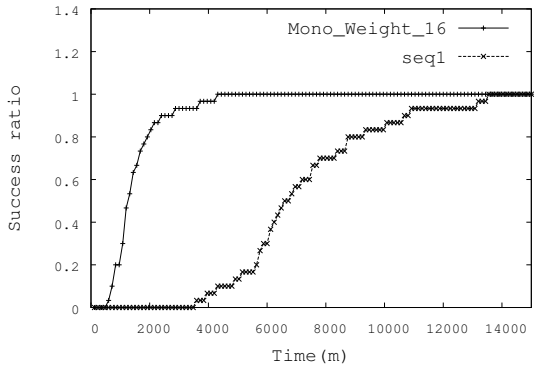


Figure 7.3: RLDs of seq1 and MONO_WEIGHT$_{16}$

The worst single parallel execution performed better than the best single sequential execution, reflecting the clear superiority of the parallel approach.

Since the parallel executions use more computational resources than the sequential ones, the improvement achieved by the parallel model must be measured. In order to measure the improvement of the parallel approach, a second analysis that considers the run-time behaviour of the sequential and parallel models has been carried out. The sequential configurations and the parallel model were executed using as finalisation condition the achievement of a certain level of quality: the mean achieved by the best sequential configuration in the first experiment. Since some of the configurations are not able to reach such a quality level, a second stopping criterion - the execution of a maximum time of 6 hours - was also considered. Figure 7.3 shows the RLDs for MONO_WEIGHT$_{16}$ and for the best behaved sequential configuration. It shows the superiority of the MONO_WEIGHT$_{16}$ model when compared to "seq1". More than half of the configurations were not able to obtain a success ratio greater than 10% for the considered quality value. This indicates that the speedup of the parallel model with respect to such configurations is very large. In order to calculate the exact value much longer executions would be needed for such bad-behaved configurations. Among the 14 configurations which have a success ratio greater than 10%, the configuration "seq1", "seq7" and 'seq14" were selected to make a deeper analysis. Table 7.1 shows the success ratio and the speedup factors to attain a 50% of success ratio. MONO_WEIGHT$_{16}$ have obtained a 100% of success ratio and a 5.09 speedup when compared with the best sequential configuration. Although linear speedup is not achieved, it must be taken into account that when solving a problem, users don't know a priori the best configuration, so the time saving is much greater than the speedup calculated versus the best configuration. In fact, when considering

Configuration	Speedup	Success ratio(%)
seq1	5.09	100
seq7	6.9	100
seq14	13.31	83.3

Table 7.1: Speedup of the new parallel model

	MONO_WEIGHT$_{16}$		RANDOM$_{16}$	
Time	Mean	Median	Mean	Median
30 m	991126233	990615500	987718266	986855500
60 m	996036733	993478000	992013666	991733500
120 m	1004046666	1003205000	995660333	995207000

Table 7.2: Mean and median fitness for MONO_WEIGHT$_{16}$ and RANDOM$_{16}$

other low-level configurations the speedup factor highly increases.
The last experiment was performed with the aim of checking the suitability of the scoring and selection schemes. The proposed scheme was compared with a model that randomly changes the configurations executed on the islands - RANDOM$_{16}$. The involved configurations, migration scheme and local stopping criteria were similar to the ones used in MONO_WEIGHT$_{16}$. In order to provide the results with statistical confidence and detect differences between the algorithms within short and long time ranges, three different time limits were considered (30 minutes, 1 hour and 2 hours) and statistical comparisons were performed. Table 7.2 shows the mean and the median of the achieved fitness by both models for the considered times. In every case, the mean and median fitness achieved by MONO_WEIGHT$_{16}$ is greater than the ones achieved by making the random mapping. Moreover, the statistical tests show that the differences between the models have been statistically significant for every considered stopping criterion.

7.4.2 Multiobjectivised Schemes

This section is devoted to present the results obtained with a set of multiobjectivised schemes. First, an analysis of several multiobjectivised configurations is performed. Then, an analysis of the MONO_WEIGHT hyperheuristic, using as low-level strategies such a set of multiobjectivised configurations, is performed.

Sequential multiobjectivised schemes

The objective of the first experiment has been to analyse the behaviour of multiobjectivisation for the 2DPP. Specifically, the effect of the MOEA and the effect of the

Table 7.3: Original objective function for the multiobjectivised configurations - Small Instance

Name	MOEA	Multiobj.	Mean	Median	Max
SEQ1	SPEA2	DBI	$5.130 \cdot 10^8$	$5.134 \cdot 10^8$	$5.152 \cdot 10^8$
SEQ2	SPEA2	DBI-THR	$5.129 \cdot 10^8$	$5.129 \cdot 10^8$	$5.157 \cdot 10^8$
SEQ3	NSGA2	ADI	$5.126 \cdot 10^8$	$5.125 \cdot 10^8$	$5.152 \cdot 10^8$
SEQ4	NSGA2	DCN	$5.124 \cdot 10^8$	$5.127 \cdot 10^8$	$5.142 \cdot 10^8$
SEQ5	SPEA2	DCN	$5.120 \cdot 10^8$	$5.121 \cdot 10^8$	$5.145 \cdot 10^8$
SEQ6	SPEA2	DCN-THR	$5.120 \cdot 10^8$	$5.118 \cdot 10^8$	$5.137 \cdot 10^8$
SEQ7	NSGA2	DBI-THR	$5.118 \cdot 10^8$	$5.119 \cdot 10^8$	$5.143 \cdot 10^8$
SEQ8	NSGA2	DCN-THR	$5.118 \cdot 10^8$	$5.115 \cdot 10^8$	$5.146 \cdot 10^8$
SEQ9	SPEA2	ADI	$5.117 \cdot 10^8$	$5.112 \cdot 10^8$	$5.144 \cdot 10^8$
SEQ10	NSGA2	DBI	$5.117 \cdot 10^8$	$5.119 \cdot 10^8$	$5.139 \cdot 10^8$
SEQ11	NSGA2	Dependent	$5.105 \cdot 10^8$	$5.102 \cdot 10^8$	$5.149 \cdot 10^8$
SEQ12	SPEA2	Dependent	$5.104 \cdot 10^8$	$5.105 \cdot 10^8$	$5.133 \cdot 10^8$
SEQ13	NSGA2	Random	$5.103 \cdot 10^8$	$5.103 \cdot 10^8$	$5.131 \cdot 10^8$
SEQ14	SPEA2	Random	$5.099 \cdot 10^8$	$5.097 \cdot 10^8$	$5.126 \cdot 10^8$
SEQ15	NSGA2	Inversion	$5.095 \cdot 10^8$	$5.093 \cdot 10^8$	$5.127 \cdot 10^8$
SEQ16	SPEA2	Inversion	$5.095 \cdot 10^8$	$5.097 \cdot 10^8$	$5.117 \cdot 10^8$

considered multiobjectivisation approaches over the quality of the obtained solutions has been studied. Moreover, additional analyses with the aim of discovering whether the most suitable approach depends on the considered instance of the 2DPP or not have been carried out. For doing that, 16 different multiobjectivised configurations have been defined. They have been obtained by combining two different MOEAs (NSGA-II and SPEA2), with the 8 multiobjectivisation schemes previously described. In every configuration, the population and the archive sizes have been fixed to $N = 10$ individuals. For the multiobjectivisation approaches that incorporates the usage of a threshold value, it has been fixed to $th = 0.99$. The variation stage has been based on using the best operator found in the previous analysis. They were the SSX operator with $p_c = 1$, and the UMD with $min\_p_m = 0.1$ and $max\_p_m = 0.15$. The stopping criterion has been fixed to 5 hours for the small instance and to 11.5 hours for the contest instance.

Table 7.3 shows, for the small instance and for each tested MA, the mean, the median, and the maximum of the achieved original objective values. The configurations have been sorted in terms of their mean values. An index based on such an order

Table 7.4: Original objective function for the multiobjectivised configurations - Contest Instance

Name	MOEA	Multiobj.	Mean	Median	Max
SEQ1	NSGA2	DCN-THR	$1.008 \cdot 10^9$	$1.007 \cdot 10^9$	$1.017 \cdot 10^9$
SEQ2	SPEA2	DCN-THR	$1.007 \cdot 10^9$	$1.006 \cdot 10^9$	$1.015 \cdot 10^9$
SEQ3	SPEA2	DBI-THR	$1.006 \cdot 10^9$	$1.007 \cdot 10^9$	$1.015 \cdot 10^9$
SEQ4	NSGA2	DBI-THR	$1.005 \cdot 10^9$	$1.005 \cdot 10^9$	$1.015 \cdot 10^9$
SEQ5	SPEA2	DCN	$1.004 \cdot 10^9$	$1.005 \cdot 10^9$	$1.013 \cdot 10^9$
SEQ6	NSGA2	DCN	$1.004 \cdot 10^9$	$1.005 \cdot 10^9$	$1.015 \cdot 10^9$
SEQ7	NSGA2	ADI	$1.004 \cdot 10^9$	$1.002 \cdot 10^9$	$1.015 \cdot 10^9$
SEQ8	SPEA2	ADI	$1.002 \cdot 10^9$	$1.002 \cdot 10^9$	$1.013 \cdot 10^9$
SEQ9	NSGA2	DBI	$1.001 \cdot 10^9$	$1.001 \cdot 10^9$	$1.011 \cdot 10^9$
SEQ10	SPEA2	DBI	$9.997 \cdot 10^8$	$9.992 \cdot 10^8$	$1.009 \cdot 10^9$
SEQ11	SPEA2	Random	$9.961 \cdot 10^8$	$9.995 \cdot 10^8$	$1.004 \cdot 10^9$
SEQ12	NSGA2	Random	$9.950 \cdot 10^8$	$9.945 \cdot 10^8$	$1.004 \cdot 10^9$
SEQ13	NSGA2	Dependent	$9.946 \cdot 10^8$	$9.952 \cdot 10^8$	$1.001 \cdot 10^9$
SEQ14	SPEA2	Dependent	$9.946 \cdot 10^8$	$9.956 \cdot 10^8$	$1.001 \cdot 10^9$
SEQ15	SPEA2	Inversion	$9.864 \cdot 10^8$	$9.861 \cdot 10^8$	$9.938 \cdot 10^8$
SEQ16	NSGA2	Inversion	$9.851 \cdot 10^8$	$9.850 \cdot 10^8$	$9.923 \cdot 10^8$

has been assigned to each configuration. From now on the first configuration, i.e. the one which has achieved the highest mean of the original objective value, will be referred to as SEQ1, while the last one will be referred to as SEQ16. It can be observed that differences among the configurations have been noticeable, showing the importance of correctly selecting the appropriate one. In fact, differences among SEQ1 and the rest of the configurations have been statistically significant, except for the configurations SEQ2 – SEQ4. Statistical tests have also confirmed that the applied multiobjectivisation approach as well as the applied MOEA have really affected the obtained results. For instance, SEQ1 has been significantly different from SEQ10. Such configurations are based on the same multiobjectivisation approach, but they consider a different MOEA. By this way, it can be noted that the proper selection of the MOEA has really affected the obtained results. Similarly, SEQ1 and SEQ5, which are both based on the SPEA2 have been statistically different. Since they only differ on the multiobjectivisation approach, the importance of properly selecting the multiobjectivisation method has also been demonstrated. On the other hand, statistical tests have shown that the incorporation of a threshold value

in the tested multiobjectivisation approaches has not affected the obtained results. The configurations that have applied a multiobjectivisation approach with threshold have not been statistically different from their homonyms without threshold. For example, the differences between the results obtained by SEQ1 and SEQ2 have not been statistically significant.

Table 7.4 shows the same information for the contest instance. In this case, noticeable differences among the considered configurations have also appeared. The results obtained by SEQ1 have been statistically different from the ones obtained by the other configurations, apart from the configurations SEQ2 and SEQ3. In addition, it has also been demonstrated by the statistical analysis that changing the applied MOEA has not produced significant differences on the obtained results. For example, differences between SEQ1 and SEQ2 have not been statistically significant. In this case, both configurations have applied the same multiobjectivisation method and have used a different MOEA. It is worthy to mention that this has happened for every pair of configurations in which the applied multiobjectivisation approach has been the same and the applied MOEA has been different. However, the statistical analysis has supported that the applied multiobjectivisation approach has really affected the quality of the obtained solutions. For instance, SEQ1 has been statistically different from SEQ4, and they have applied the DCN-THR and the DBI-THR multiobjectivisations, respectively. Finally, it has been statistically demonstrated that the incorporation of a threshold value in the applied multiobjectivisation methods has an effect over the obtained results. Only SEQ5 and the corresponding configuration which has applied a multiobjectivisation approach with threshold (SEQ2) have not presented significant statistical differences. In the rest of the cases, the configurations that have applied a multiobjectivisation approach with threshold have been statistically different from their homonyms without threshold.

Figure 7.4 shows, for both instances, the evolution of the mean of the original objective value for several configurations. Once more it can be observed that differences among the results obtained by the tested configurations are considerable. Although in some executions stagnation in local optima has appeared, the mean and the median of the original objective value have kept increasing even after a large amount of time. In fact, multiobjectivisation has been able to avoid premature convergence in almost all cases. Considering both tested instances, it is important to remark that the most suitable configurations have been different. For example, the configuration SEQ1 for the small instance has been the configuration SEQ10 for the contest instance. Similarly, the configuration SEQ1 for the contest instance has been the configuration SEQ8 for the first one. By this way, the most suitable configurations depend on the features of the considered instance. Given a new instance, it is difficult to predict which configuration will provide the best results. In addition, if the

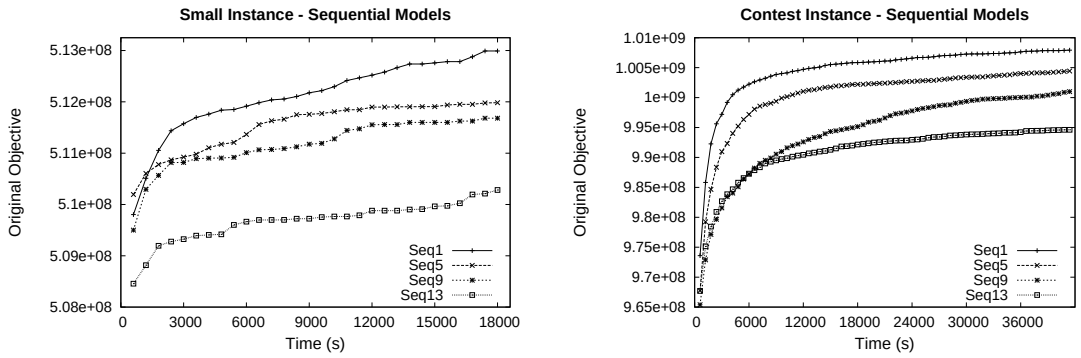


Figure 7.4: Evolution of the mean of the original objective function obtained with the multiobjectivised schemes

number of considered configurations is very large, testing each one of them might be unfeasible. Therefore, the application of a hyperheuristic seems very promising. Finally, since the sequential models have not converged even after a very large amount of time, the usage of parallel models also seems a promising approach.

Hyperheuristic for multiobjectivised schemes

The objective of the second experiment has been to analyse the behaviour of the MONO_WEIGHT hyperheuristic when it is used with multiobjectivised low-level configurations. Given the importance of the migrations, it has been tested with four migration stages. They were generated by combining two topologies with two replacement selectors. The tested topologies were the RING and the ALL. The considered replacement schemes were the ELI and the HAM. In every case, an *elitist migration scheme* is applied. Specifically, a subpopulation individual is migrated when its cost is lower than the cost of any member of its previous generation. This is checked in every generation. The following nomenclature is used to identify the different migration stages: *Replacement_Scheme-Topology*. A total number of $n_p = 4$ islands have been considered. The global stopping criterion has been fixed to 5 hours of execution for the small instance, while for the contest instance it has been fixed to 11.5 hours. For both instances, the local stopping criterion has been fixed to 10 minutes. The MONO_WEIGHT model has been applied with an adaptation level $k = 10$, and the value of β has been fixed in a way that a 10% of the decisions performed by the hyperheuristic follow a uniform distribution, i.e. $\beta \cdot n_h = 0.1$. Finally, the 16 multiobjectivised configurations previously analysed have been used as low-level configurations.

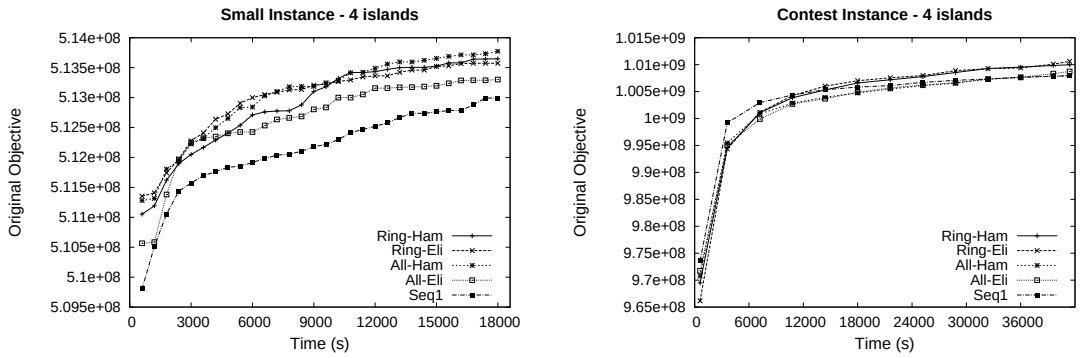


Figure 7.5: Evolution of the mean of the original objective function obtained with the MONO_WEIGHT model with 4 worker islands

Figure 7.5 shows, for both considered instances, the evolution of the mean of the original objective value for the MONO_WEIGHT model executed with the four considered migration stages. In order to compare the results obtained by the parallel models, it also shows the data of the best sequential configuration (SEQ1). For both instances, the parallel models have been able to achieve a higher mean of the original objective value than the best sequential approach. Moreover, the differences among the parallel model which has obtained the highest mean value and the best sequential approach, for each considered instance, have been statistically significant. In the case of the small instance, such a parallel approach has applied the migration stage ALL-HAM, while for the contest one the parallel model which has obtained the highest mean value has used the migration stage RING-ELI.

Figure 7.6 show the boxplots of the MONO_WEIGHT model executed with the different migration stages for the small and contest instances. They also show the boxplot of the corresponding best sequential configuration for each instance. In the case of the small instance, differences among the parallel approaches are not very noticeable. However, they are slightly superior to the best sequential approach. In the case of the contest instance, the hyperheuristic with the migration stage RING-ELI is the best-behaved. Differences between the parallel model which has applied the migration stage RING-ELI and the parallel model using the migration stage ALL-HAM have been statistically significant.

Finally, it is worthy to mention that by using the parallel model with four processors it is not necessary to separately test every considered configuration. By this way, high quality solutions can be achieved by a single execution of the MONO_WEIGHT model. Therefore, a larger amount of computational resources can be saved.

An additional experiment devoted to analyse the effect caused by the migration

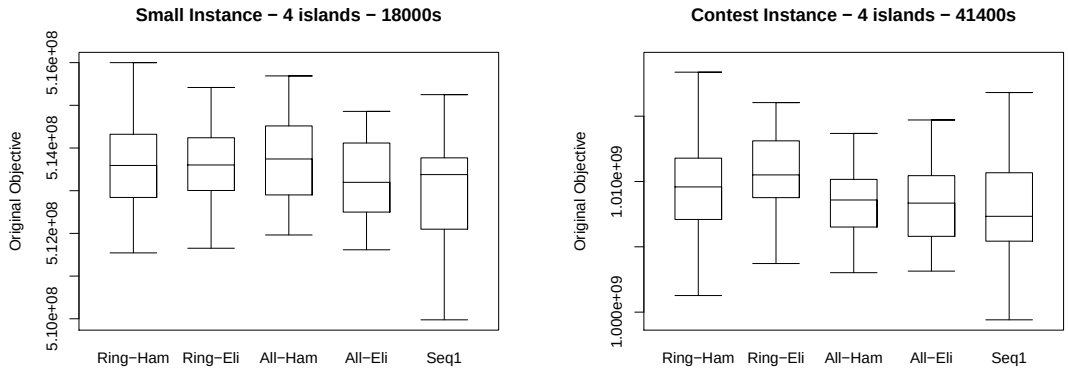


Figure 7.6: Boxplots of the obtained fitness with the MONO_WEIGHT model with 4 islands

Table 7.5: Statistical tests of the MONO_WEIGHT model - 16 islands - 5 hours - Small Instance

	Ring-Ham	Ring-Eli	All-Ham	All-Eli
Ring-Ham	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$
Ring-Eli	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$
All-Ham	$\downarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$
All-Eli	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$

stage and its relationship with the number of considered islands has been carried out. In this case, the MONO_WEIGHT model has been executed with the same parameterisation as in the previous experiment, but using a total number of 8, 16, and 32 worker islands (n_p).

Taking into consideration the small instance, statistical differences among the different migration stages have not been significant when the MONO_WEIGHT model has been applied with 4 and 8 worker islands. However, when the MONO_WEIGHT model has been executed with 16 and 32 islands, statistical differences among the considered migration stages have appeared. This means that the proper selection of the migration stage is more important when the number of considered worker islands is higher. Tables 7.5 and 7.6 show the statistical significances for the different migration stages considering 16 and 32 worker islands, respectively. Every cell shows if the row model is statistically better ($\uparrow$), not different ($\leftrightarrow$), or worse($\downarrow$) than the corresponding column model. It can be observed that the MONO_WEIGHT model which has applied the migration stage ALL-HAM has provided the worst behaviour.

Table 7.6: Statistical tests of the MONO_WEIGHT model - 32 islands - 5 hours - Small Instance

	Ring-Ham	Ring-Eli	All-Ham	All-Eli
Ring-Ham	↔	↔	↑	↔
Ring-Eli	↔	↔	↔	↔
All-Ham	↓	↔	↔	↔
All-Eli	↔	↔	↔	↔

Table 7.7: Statistical tests of the MONO_WEIGHT model - 4 islands - 11.5 hours - Contest Instance

	Ring-Ham	Ring-Eli	All-Ham	All-Eli
Ring-Ham	↔	↔	↔	↔
Ring-Eli	↔	↔	↑	↔
All-Ham	↔	↓	↔	↔
All-Eli	↔	↔	↔	↔

Table 7.8: Statistical tests of the MONO_WEIGHT model - 8, 16, 32 islands - 11.5 hours - Contest Instance

	Ring-Ham	Ring-Eli	All-Ham	All-Eli
Ring-Ham	↔	↔	↑	↑
Ring-Eli	↔	↔	↑	↑
All-Ham	↓	↓	↔	↔
All-Eli	↓	↓	↔	↔

In the case of the contest instance, Table 7.7 shows the results of the statistical tests for the different migration stages considering 4 worker islands. Similarly, Table 7.8 shows the same information when 8, 16, and 32 islands have been used. The amount of significant statistical differences has been larger when 8, 16, and 32 islands have been applied. This confirms that, as in the small instance, the importance of selecting the appropriate migration stage increases when a higher amount of worker islands are considered. Moreover, except for 4 worker islands, the results obtained by the models which have applied the migration topology ALL have been statistically worse than the results obtained by the models which have made use of the migration topology RING.

In order to better analyse the importance of selecting the appropriate migration stage for the MONO_WEIGHT model, another analysis has been carried out. Considering the mean of the original objective value achieved by the parallel models with 32 islands, the best and worst of them have been selected for each instance. Figure 7.7

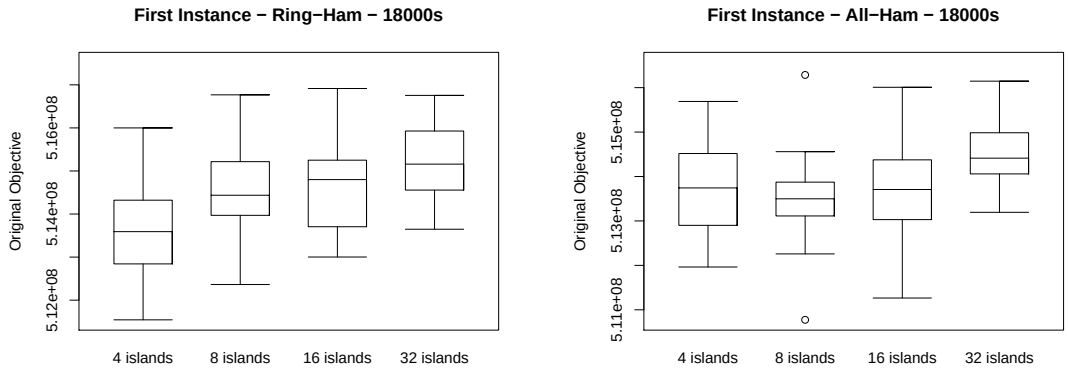


Figure 7.7: Boxplots of the MONO_WEIGHT model with the best and worst migration stages for the Small Instance

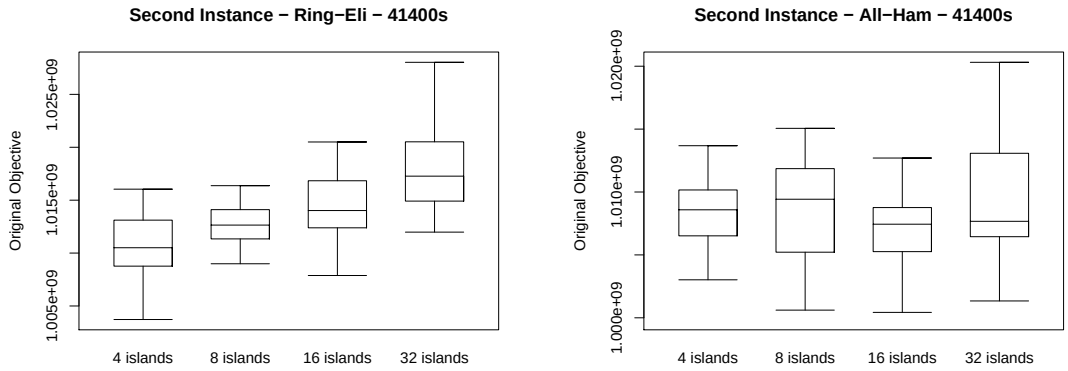


Figure 7.8: Boxplots of the MONO_WEIGHT model with the best and worst migration stage for the Contest Instance

shows, for the small instance, the boxplots of the best and worst parallel models when they have been run with up to 32 islands. The same information is shown in Figure 7.8 for the contest instance. For both instances, it can be observed that the trend towards obtaining better objective values as the number of islands increases is clear when the best migration stage has been considered. However, this has not occurred with the worst migration stage. Finally, it is worthy to mention that differences between the quality of the solutions obtained by the best and worst approaches have been considerable.

The previous analysis has compared different parallel models in terms of the quality achieved at fixed times. However, it is important to quantify the improvement achieved by such parallel approaches in terms of the saved time. For doing it, an additional study based on the usage of the RLDs has been performed. Figure 7.9 shows, for the small instance, the RLDs of the best and worst parallel models with up to 32 worker islands, respectively. They also include the RLDs of the sequential configurations SEQ1 and SEQ3 so as to compare the results obtained by the parallel models. The same information is shown in Figure 7.10 for the contest instance. In order to calculate the RLDs for both instances, the quality level has been fixed as the median of the original objective value achieved by the configuration SEQ3. In the case of the small instance, the parallel models which have applied the best migration stage have clearly outperformed the sequential configurations, obtaining higher or the same success ratios in a lower amount of time. In addition, not only high quality solutions have been achieved by the best parallel model, but also such solutions have been obtained in less time when the number of worker islands has been increased. For example, the best parallel models with 16 and 32 worker islands have been able to achieve a 100% of success ratio, i.e. every execution has reached the fixed quality level. However, the best parallel model with 32 islands has obtained such a success ratio in a lower amount of time. Similar conclusions can be extracted in the case of the parallel model that has used the worst-behaved migration stage. However, it has obtained solutions with a lower quality level than the ones obtained by the best parallel model at the end of the executions, as it has been observed in the previous analysis. Moreover, none of the worst parallel models has been able to reach a 100% of success ratio. Therefore, selecting the appropriate migration stage not only affects the quality of the obtained solutions, but also the total amount of time and processors required to achieve a fixed quality level.

In the case of the contest instance, the parallel model that has applied the best migration stage with 4 worker islands has behaved similarly than the best sequential configuration when low success ratios have been considered. Nevertheless, when the success ratio has risen, differences between both models have become greater in favour of the parallel model. Taking into consideration the best parallel models with 8, 16, and 32 worker islands the benefits of adding worker islands have been noticeable. The different success ratios have been achieved in less time when a higher amount of islands has been used. In addition, the best parallel models with 16 and 32 islands have been able to achieve a 100% of success ratio. On the other hand, it can be observed that the behaviour of the parallel model which has applied the ALL-HAM migration stage has been poor. For example, such a model with 4 worker islands has obtained certain values of the success ratio in a lower amount of time than the same model using 32 islands. In addition, as in the case of the

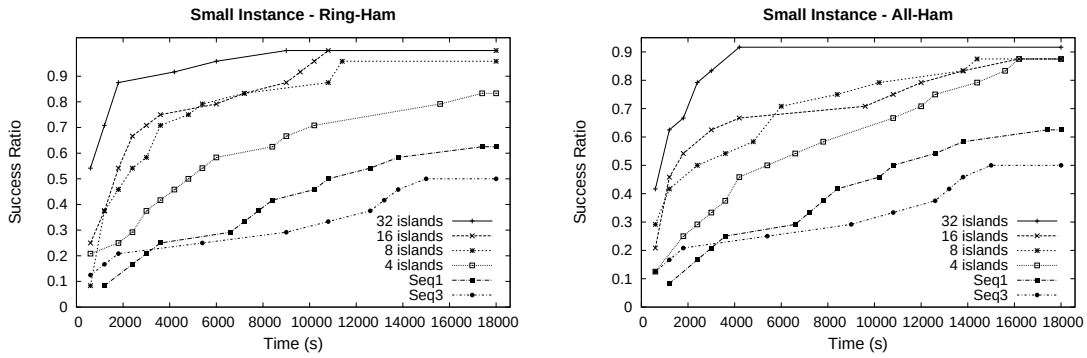


Figure 7.9: RLDs of the MONO_WEIGHT model with the best and worst migration stages for the Small Instance

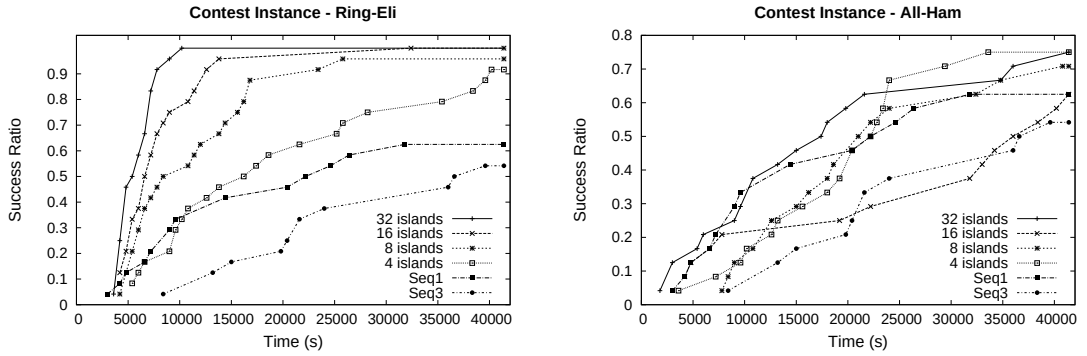


Figure 7.10: RLDs of the MONO_WEIGHT model with the best and worst migration stages for the Contest Instance

small instance, none of the parallel models using the worst-behaved migration stage have achieved a 100% of success ratio. Consequently, increasing the number of processors has not provided any advantage when the worst migration stage has been applied. Finally, it is worthy to mention that for this particular instance, changing the most suitable migration stage has clearly affected the behaviour of the whole MONO_WEIGHT model, since differences between the best and the worst migration stages have been more noticeable than for the small instance.

In order to quantify the effects that the migration stage has caused over the scalability of the MONO_WEIGHT model, speedup factors have been calculated using the data provided by the RLDs. Table 7.9 shows the speedup factors, with regard to SEQ1, obtained by the MONO_WEIGHT model when it has been applied with the corresponding best and worst migration stages to the small instance. In order to

Table 7.9: Speedup factors of MONO_WEIGHT with the best and worst migration stages - Small Instance

	Best	Worst
4 islands	1.71	1.61
8 islands	3.98	1.07
16 islands	6.29	1.21
32 islands	15.73	7.26

Table 7.10: Speedup factors of MONO_WEIGHT with the best and worst migration stages - Contest Instance

	Best	Worst
4 islands	2.3	1.64
8 islands	4.10	2.57
16 islands	5.70	1.36
32 islands	18.81	2.90

calculate such factors the following steps have been performed. Firstly, given a model with n_p worker islands, a relative speedup factor ($spr_{[n_p]}$) has been calculated with regard to the model with $n_p \div 2$ worker islands. In the case of the parallel models with $n_p = 4$ islands, the best sequential configuration (SEQ1) has been used as reference. For this instance, and for each relative speedup factor, the quality level has been fixed as the lowest median of the original objective value achieved by both considered models in 5 hours. The relative speedup is given by the division between the time invested by the model using a lower number of processors and the time invested by the model using a higher amount of processors. Such times have been obtained by considering a 50% of success ratio. Once the relative speedup factors have been calculated, the resultant speedup factor for the model that has applied n_p processors ($sp_{[n_p]}$) is calculated as follows:

$$sp_{[n_p]} = \begin{cases} spr_{[n_p]} \cdot sp_{[n_p \div 2]} & if \quad n_p \neq 4 \\ spr_{[4]} & if \quad n_p = 4 \end{cases} \tag{7.4}$$

The resultant speedup factors of the contest instance are shown in Table 7.10. The procedure to calculate them has been the same as for the small instance, but a time equal to 11.5 hours has been taken into account.

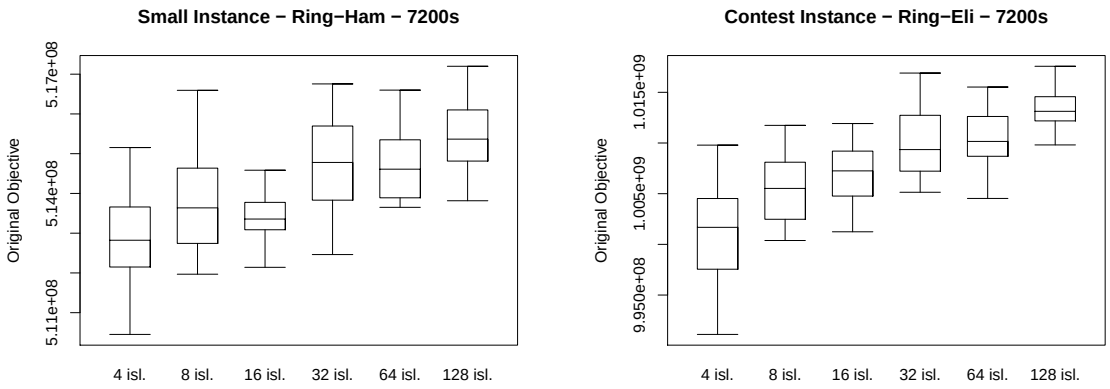


Figure 7.11: Boxplots of the MONO_WEIGHT model with the best migration stage with 64 and 128 islands

For both instances, it can be observed that the speedup factors have increased when the best parallel model has been applied with a larger amount of worker islands. For example, in the case of the small instance, the best parallel model with 16 worker islands has obtained a speedup factor equal to 6.29, while the same model considering 32 islands has achieved a speedup factor equal to 15.73. In this case, the relative speedup factor calculated for both models has been greater than one. This means that the model with 32 islands has achieved the fixed quality level in a 50% of the executions in less time than the model with 16 worker islands. However, this fact has not happened when the corresponding worst parallel model has been applied to each instance. For example, in the case of the contest instance, the worst parallel model with 8 worker islands has obtained a speedup factor equal to 2.57 while the worst parallel model with 16 worker islands has obtained a speedup factor equal to 1.36. In this case, the relative speedup factor calculated for both models has been lower than one. This means that the model with a lower number of worker islands has achieved the fixed quality level in a 50% of the executions in less time than the model which has considered a larger amount of islands. Therefore, incorporating a larger amount of processors in the worst-behaved parallel model for each considered instance has not provided good results.

Finally, the MONO_WEIGHT model using the best-behaved migration stage for each considered instance has been executed with 64 and 128 worker islands. The parameterisation of the MONO_WEIGHT model has been the same than the one applied in the previous experiments. However, the global stopping criterion has been fixed to 2 hours due to availability of computational resources.

Table 7.11: Speedup factors of MONO_WEIGHT with the best migration stage for both instances

	64 islands	128 islands
Small instance	12.58	41.9
Contest instance	20.51	25.02

The boxplots of the MONO_WEIGHT model with up to 128 worker islands applying the corresponding best-behaved migration stage for each instance are shown in Figure 7.11. Even with a so large number of worker islands, the quality of the solutions obtained by MONO_WEIGHT has kept increasing as more resources have been considered. In general, the larger the number of worker islands, the higher the quality of the obtained solutions. Nevertheless, some scalability problems have appeared when the best parallel model has been applied to the small instance with 16 and 64 worker islands. In the case of the contest instance, problems have only appeared with 64 islands, but it is worthy to mention that the median of the original objective value has improved with every increase of the number of processors.
Table 7.11 shows the speedup factors, with regard to SEQ1, of the parallel model using the best migration stage for each instance. Speedup factors have been obtained following the same procedure than in the previous experiment. In order to obtain the relative speedup factors for both instances, the quality level has been fixed as the lowest median of the fitness of both considered models in 2 hours. The calculated speedup factors have confirmed the benefits of adding a larger amount of processors. In the case of the small instance, such benefits have been more noticeable.
The executions with $n_p = 128$ islands have been able to provide better results than the best previously known solution for both instances. In the small instance the value 517.199.441 has been obtained. In the case of the contest instance a larger execution which considered a stopping criterion of 15 days was carried out. The obtained solution - which is the best-known solution - has the value 1.038.329.890.

Part IV

Conclusions

Conclusions

Metaheuristics have proven to be adequate techniques for solving complex optimisation problems. However, solving novel problem and instances requires a large computational and user effort. This research deals with the techniques that facilitate the usage of metaheuristics. Specifically, a set of hyperheuristics for multi-point metaheuristics have been proposed. The defined hyperheuristics are an extension of the choice-function based schemes. Initially, several schemes were proposed. The ones which obtained high-quality solutions have been deeply analysed. Several hyperheuristic for mono-objective and multi-objective optimisation have been proposed. They are based on assigning more resources to the configurations that have performed better in previous stages of the optimisation.
In addition, some of the last innovations of optimisation have been analysed. Specifically, several novel multiobjectivisation schemes have been proposed. The novel schemes are based on considering the diversity and the quality of the solutions simultaneously. The optimal usage of such multiobjectivisations requires the specification of an additional parameter. In order to facilitate their usage, they have been integrated with the designed hyperheuristics.
The validation of the proposals was initially performed with some well-known mono-objective and multi-objective optimisation problems. It has been shown that by properly configuring the hyperheuristics and multiobjectivisations high-quality solutions can be found. In some cases, the time required to converge to high-quality solutions is larger than the time used by the best considered low-level configurations. In such cases the advantage is that the manual testing of each low-level configuration can be avoided. In addition, some cases in which the hyperheuristic obtains better solutions than any of the single low-level configurations have been found. The reason is that there are some cases in which the relative performance between the low-level configurations depends on the optimisation stage. Thus, by combining the advantages of each low-level approach better results can be obtained.
Regarding the parallel approaches, several scalability analyses have been performed. Adequate speedup factors have been obtained when few processors have been used. However, when a large amount of processors have been used, the obtained speedup

factors have highly depended on the tackled problem and on the migration stages. Thus, for large systems it is important to set up a proper migration stage.
Several complex and practical optimisation problems have also been addressed. Specifically, three problems that arise in the communication field have been studied. In such cases, several novel schemes have been proposed, and they have been integrated with the proposed hyperheuristics and multiobjectivisation schemes. The obtained results confirm the suitability of the proposed schemes. In several instances, the best-known results have been obtained with the new proposals. In addition, a two-dimensional packing problem which was proposed in a contest of the GECCO has been solved. A tailor-made neighbourhood definition was proposed and integrated in an evolutionary approach that was controlled by the designed hyperheuristics. Multiobjectivisation was also successfully applied to such a problem. The best-known results have been obtained with the designed schemes.
Therefore, the proposed algorithmic methods have proven to be effective for a large number of optimisation problems. It has been shown that the quality of the results depends on some of the parameters of the methods. Moreover, the most adequate parameters of the hyperheuristics depend on the problem and instances. However, it is much easier to configure the designed methods, that to configure the low-level schemes. In fact, in most of the cases, competent results have been obtained independently of the tested parameters. In addition, since the usage of parallel environments has been enabled with the designed methods, the time invested for obtaining high-quality results has been highly reduced in many cases.
Finally, it is worthy to mention that a new tool that enables the cooperation of metaheuristics and their usage in sequential and parallel environments has been developed. The tool allows executing the models developed in this research, as well as many other schemes designed by other researchers. The functionalities of the tool can be extended through the definition of plugins, while the kind of execution to perform can be specified by using a configuration file. This means that it is not required to know the internals of the tool to use it.
Several lines of future work can be studied. First, some of the ideas that were used to design some of the hyperheuristics that did not obtain high-quality results might be carefully analysed. For instance, we believe that the hyperheuristics that are based on applying an exponential regression to estimate future results might be very useful. The direct usage of regressions is not adequate because the stochastic behaviour of the low-level configurations produce several drawbacks. However, we believe that it is a promising line for future work. In addition, the designed hyperheuristics does not distinguish between symbolic and numeric parameters. Since numeric parameters have a structure where relations between different values can be exploited, some best-suited hyperheuristics might be defined for such cases.

Part V

Appendices

Appendix

A

List of Publications

This appendix presents the publications that have been produced as a result of the research conducted over the course of this PhD. These publications include book chapters, articles in international journals of relevance to the particular research field, and contributions to international conferences with review committees for ensuring the quality and validity of the selected works.

Book Chapters

[1] C. Segura, E. Segredo, and C. León. *Evolve - A Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation II*, chapter Analysing the Robustness of Multiobjectivisation Approaches Applied to Large Scale Optimisation Problems, pages 365 – 391. Advances in Intelligent Systems and Computing. Springer-Verlag, 2012.

International Journals

[1] J. de Armas, C. Leon, G. Miranda, and C. Segura. Optimisation of a Multi-Objective Two-Dimensional Strip Packing Problem based on Evolutionary Algorithms. *International Journal of Production Research*, 48(07):2011–2028, Feb. 2010.

[2] J. de Armas, C. León, G. Miranda, and C. Segura. Remote service to solve the two-dimensional cutting stock problem: an application to the Canary Islands

costume. *International Journal of Grid and Web Services*, 4:342–351, December 2008.

[3] C. León, G. Miranda, and C. Segura. METCO: A Parallel Plugin-Based Framework for Multi-Objective Optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–588, 2009.

[4] F. Luna, C. Estébanez, C. León, J. Chaves-González, A. Nebro, R. Aler, C. Segura, M. Vega-Rodríguez, E. Alba, J. Valls, G. Miranda, and J. Gómez-Pulido. Optimization algorithms for large-scale real-world instances of the frequency assignment problem. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 15:975–990, 2011.

[5] S. P. Mendes, G. Molina, M. A. Vega-Rodríguez, J. A. Gómez-Pulido, Y. Sáez, G. Miranda, C. Segura, E. Alba, P. Isasi, C. León, and J. M. Sánchez-Pérez. Benchmarking a wide spectrum of metaheuristic techniques for the radio network design problem. *IEEE Transactions on Evolutionary Computation*, 13(5):1133–1150, Oct. 2009.

[6] C. Segura, G. Miranda, and C. León. Parallel Hyperheuristics for the Frequency Assignment Problem. *Memetic Computing*, 3(1):33–49, 2010.

International Conferences

[1] E. Alba, A. Cervantes, J. Gómez, P. Isasi, M. Jaraíz, C. León, C. Luque, F. Luna, G. Miranda, A. Nebro, R. Pérez, and C. Segura. Metaheuristics approaches for optimal broadcasting design in metropolitan MANETs. In *Eleventh International Conference on Computer Aided Systems Theory (EUROCAST 2007)*, 2007.

[2] J. d. Armas, C. León, G. Miranda, and C. Segura. Remote Service to Solve the Two-Dimensional Cutting Stock Problem: An Application to the Canary Islands Costume. In *Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems*, CISIS '08, pages 971–976. IEEE Computer Society, 2008.

[3] O. Gonzalez, C. León, G. Miranda, C. Rodríguez, and C. Segura. A parallel skeleton for the strength pareto evolutionary algorithm 2. In *15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007)*, pages 434–441. IEEE Computer Society, 2007.

[4] C. Leon, G. Miranda, C. Rodriguez, and C. Segura. A Distributed Parallel Algorithm to Solve the 2D Cutting Stock Problem. In *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 429–434. IEEE Computer Society, 2008.

[5] C. Leon, G. Miranda, E. Segredo, and C. Segura. Parallel Library of Multi-objective Evolutionary Algorithms. *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 28–35, 2009.

[6] C. León, G. Miranda, and C. Segura. Parallel skeleton for multi-objective optimization. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 906–906. ACM, 2007.

[7] C. Leon, G. Miranda, and C. Segura. Optimizing the Configuration of a Broadcast Protocol through Parallel Cooperation of Multi-objective Evolutionary Algorithms. In *Advanced Engineering Computing and Applications in Sciences, 2008. ADVCOMP '08. The Second International Conference on*, pages 135 – 140, oct. 2008.

[8] C. Leon, G. Miranda, and C. Segura. Parallel hyperheuristic: a self-adaptive island-based model for multi-objective optimization. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 757–758. ACM, 2008.

[9] C. Leon, G. Miranda, and C. Segura. A memetic algorithm and a parallel hyperheuristic island-based model for a 2d packing problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1371–1378. ACM, 2009.

[10] C. León, G. Miranda, C. Rodríguez, and C. Segura. 2D Cutting Stock Problem: A New Parallel Algorithm and Bounds. In A.-M. Kermarrec, L. Bougé, and T. Priol, editors, *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 795–804. Springer Berlin / Heidelberg, 2007.

[11] C. León, G. Miranda, E. Segredo, and C. Segura. Parallel Hypervolume-Guided Hyperheuristic for Adapting the Multi-objective Evolutionary Island Model. In N. Krasnogor, M. Melián-Batista, J. Pérez, J. Moreno-Vega, and D. Pelta, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008)*, volume 236 of *Studies in Computational Intelligence*, pages 261–272. Springer Berlin / Heidelberg, 2009.

[12] C. León, G. Miranda, and C. Segura. Optimizing the Broadcast in MANETs Using a Team of Evolutionary Algorithms. In I. Lirkov, S. Margenov, and J. Wasniewski, editors, *Large-Scale Scientific Computing*, volume 4818 of *Lecture Notes in Computer Science*, pages 569–576. Springer Berlin / Heidelberg, 2008.

[13] C. León, G. Miranda, and C. Segura. A Parallel Plugin-Based Framework for Multi-objective Optimization. In J. Corchado, S. Rodríguez, J. Llinas, and J. Molina, editors, *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50 of *Advances in Soft Computing*, pages 142–151. Springer Berlin / Heidelberg, 2009.

[14] C. León, G. Miranda, and C. Segura. Hyperheuristics for a dynamic-mapped multi-objective island-based model. In S. Omatu, M. Rocha, J. Bravo, F. Fernández, E. Corchado, A. Bustillo, and J. Corchado, editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, volume 5518 of *Lecture Notes in Computer Science*, pages 41–49. Springer Berlin Heidelberg, 2009.

[15] F. Luna, C. Estébanez, C. León, J. M. Chaves-González, E. Alba, R. Aler, C. Segura, M. A. Vega-Rodríguez, A. J. Nebro, J. M. Valls, G. Miranda, and J. A. Gómez-Pulido. Metaheuristics for solving a real-world frequency assignment problem in GSM networks. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1579–1586. ACM, 2008.

[16] G. Miranda, J. de Armas, C. Segura, and C. León. Hyperheuristic codification for the multi-objective 2d guillotine strip packing problem. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[17] E. Segredo, C. Segura, and C. Leon. A multiobjectivised memetic algorithm for the Frequency Assignment Problem. In *2011 IEEE Congress on Evolutionary Computation (CEC)*, pages 1132 –1139, june 2011.

[18] E. Segredo, C. Segura, and C. Leon. Analysing the robustness of multiobjectivisation parameters with large scale optimisation problems. In *2012 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, june 2012.

[19] E. Segredo, C. Segura, and C. León. On the comparison of parallel island-based models for the multiobjectivised antenna positioning problem. In A. König, A. Dengel, K. Hinkelmann, K. Kise, R. Howlett, and L. Jain, editors,

Knowledge-Based and Intelligent Information and Engineering Systems, volume 6881 of *Lecture Notes in Computer Science*, pages 32–41. Springer Berlin Heidelberg, 2011.

[20] E. Segredo, C. Segura, and C. León. Analysing the adaptation level of parallel hyperheuristics applied to mono-objective optimisation problems. In D. Pelta, N. Krasnogor, D. Dumitrescu, C. Chira, and R. Lung, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2011)*, volume 387 of *Studies in Computational Intelligence*, pages 169–182. Springer Berlin / Heidelberg, 2012.

[21] C. Segura, A. Cervantes, A. J. Nebro, M. D. Jaraíz-Simón, E. Segredo, S. García, F. Luna, J. A. Gómez-Pulido, G. Miranda, C. Luque, E. Alba, M. A. Vega-Rodríguez, C. León, and I. M. Galván. Optimizing the DFCN Broadcast Protocol with a Parallel Cooperative Strategy of Multi-Objective Evolutionary Algorithms. In *Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization*, EMO '09, pages 305–319, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] C. Segura, Y. Gonz'alez, G. Miranda, and C. Le'on. A multi-objective evolutionary approach for the antenna positioning problem. In R. Setchi, I. Jordanov, R. Howlett, and L. Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, volume 6276 of *Lecture Notes in Computer Science*, pages 51–60. Springer Berlin / Heidelberg, 2010.

[23] C. Segura, Y. González, G. Miranda, and C. León. Parallel Hyperheuristics for the Antenna Positioning Problem. In A. de Leon F. de Carvalho, S. Rodríguez-González, J. De Paz Santana, and J. Rodríguez, editors, *Distributed Computing and Artificial Intelligence*, volume 79 of *Advances in Intelligent and Soft Computing*, pages 471–479. Springer Berlin / Heidelberg, 2010.

[24] C. Segura, E. Segredo, Y. González, and C. León. Multiobjectivisation of the Antenna Positioning Problem. In A. Abraham, J. Corchado, S. González, and J. De Paz Santana, editors, *International Symposium on Distributed Computing and Artificial Intelligence*, volume 91 of *Advances in Intelligent and Soft Computing*, pages 319–327. Springer Berlin / Heidelberg, 2011.

[25] C. Segura, E. Segredo, and C. León. Parallel island-based multiobjectivised memetic algorithms for a 2D packing problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1611–1618. ACM, 2011.

[26] C. Segura, E. Segredo, and C. Leon. Analysing the adaptation level of parallel hyperheuristics applied to multiobjectivised benchmark problems. *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 138–145, 2012.

Bibliography

[1] K. I. Aardal, S. P. M. V. Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, 2007.

[2] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[3] H. A. Abbass and K. Deb. Searching under multi-evolutionary pressures. In *Proceedings of the Fourth Conference on Evolutionary Multi-Criterion Optimization*, pages 391–404. Springer-Verlag, 2003.

[4] M. Affenzeller and S. Wagner. Sasegasa: A new generic parallel evolutionary algorithm for achieving highest quality results. *Journal of Heuristics*, 10(3):243–267, May 2004.

[5] S. Ahmadi, R. Barone, P. Cheng, P. Cowling, and B. McCollum. Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. In *Proceedings of multidisciplinary international scheduling: theory and applications (MISTA 2003)*, pages 155 – 171, Nottingham, 1993.

[6] E. Alba. Evolutionary algorithms for optimal placement of antennae in radio network design. *Parallel and Distributed Processing Symposium, International*, 7:168, 2004.

[7] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[8] E. Alba, A. Cervantes, J. Gómez, P. Isasi, M. D. Jaraiz, C. León, C. Luque, F. Luna, G. Miranda, A. J. Nebro, R. Pérez, and C. Segura. Metaheuristic Approaches for Optimal broadcasting Design in Metropolitan MANETs.

In *11th International Conference on Computer Aided Systems Theory (EUROCAST'2007)*, volume 4739 of *LNCS*, Las Palmas de Gran Canaria, Spain, February 2007. Springer-Verlag.

[9] E. Alba, B. Dorronso, F. Luna, A. J. Nebro, P. Bouvry, and L. Hogie. A Cellular Multi-Objective Genetic Algorithm for Optimal Broadcasting Strategy in Metropolitan MANETs. *Computer Communications*, 30(4):685–697, 2007.

[10] E. Amaldi, A. Capone, F. Malucelli, and C. Mannino. Optimization problems and models for planning cellular networks. In *Handbook of Optimization in Telecommunication*, pages 917–939. Springer, 2006.

[11] I. Araya, B. Neveu, and M.-C. Riff. An Efficient Hyperheuristic for Strip-Packing Problems. In C. Cotta and K. Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 61–76. Springer, 2008.

[12] A. Avenali, C. Mannino, and A. Sassano. Minimizing the span of d-walks to compute optimum frequency assignments. *Mathematical Programming*, 91(2):357–374, 2002.

[13] M. Ayob and G. Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies (InTech'03)*, pages 132–141, Chiang Mai, Thailand, December 17-19 2003.

[14] T. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*. North-Holland, Amsterdam, 1992.

[15] T. Bäck and H. paul Schwefel. Evolutionary algorithms: Some very old strategies for optimization and adaptation. In *New Computing Techniques in Physics Research II: Proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence, and Expert Systems for High Energy and Nuclear Physics*, pages 247–254, 1992.

[16] T. Bäck, G. Rüdolph, and H. Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, 1991.

[17] M. Bader-El-Den and R. Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In N. Monmarché, E.-G. Talbi, P. Collet,

M. Schoenauer, and E. Lutton, editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin / Heidelberg, 2008.

[18] T. Baeck, D. Fogel, and Z. Michalewicz, editors. *Advanced Algorithms and Operations (Evolutionary Computation)*. Taylor & Francis, 1 edition, November 2000.

[19] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. *Parallel Iterative Algorithms: From Sequential to Grid Computing (Chapman & Hall/Crc Numerical Analy & Scient Comp. Series)*. Chapman & Hall/CRC, 2007.

[20] R. Bai. *An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, United Kingdom, 2005.

[21] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CS-94-163, Carnegie Mellon Univ., Pittsburgh, PA, 1994.

[22] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. Number CS-95-141, pages 38–46. Morgan Kaufmann Publishers, 1995.

[23] J. Bather. *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[24] E. B. Baum. Towards practical 'neural' computation for combinatorial optimization problems. In *AIP Conference Proceedings 151 on Neural Networks for Computing*, pages 53–58, Woodbury, NY, USA, 1987. American Institute of Physics Inc.

[25] J. Baxter. Local optima avoidance in depot location. *The Journal of the Operational Research Society*, 32(9):pp. 815–819, 1981.

[26] M. Biazzini, B. Banhelyi, A. Montresor, and M. Jelasity. Distributed hyper-heuristics for real parameter optimization. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1339–1346, New York, NY, USA, 2009. ACM.

[27] S. Binato, W. Hery, D. M. Loewenstern, and M. G. C. Resende. A grasp for job shop scheduling. In *Essays and Surveys on Metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2000.

[28] G. E. Blelloch and B. M. Maggs. Programming parallel algorithms. *Communications of the ACM*, 39:85–97, 1996.

[29] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[30] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, Sept. 2003.

[31] R. Borndörfer, A. Eisenblätter, M. Grötschel, and A. Martin. Frequency assignment in cellular phone networks. *Annals of Operations Research*, 76:73–93, 1998.

[32] G. Brassard and P. Bratley. *Fundamentals of Algorithms*. Prentice-Hall, New Jersey, 1996.

[33] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS J. on Computing*, 15(4):347–368, 2003.

[34] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler. Do additional objectives make a problem harder? In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 765–772, New York, NY, USA, 2007. ACM.

[35] L. Bui, H. Abbass, and J. Branke. Multiobjective optimization for dynamic environments. In *The 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2349 – 2356 Vol. 3, 2005.

[36] E. Burke, E. Burke, and G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.

[37] E. Burke, G. Kendall, J. L. Silva, R. O'Brien, and E. Soubeiga. An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 3, pages 2263–2270, Edinburgh, Scotland, September 2-5 2005.

[38] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyperheuristics: A Survey of the State of the Art. Technical Report NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham, Computer Science, 2010.

[39] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. A classification of hyper-heuristics approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 15, pages 449–468. Springer, 2nd edition, 2010.

[40] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Handbook of Meta-heuristics*. Kluwer, 2003.

[41] E. K. Burke, G. Kendall, and E. Soubeiga. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[42] E. K. Burke, J. D. Landa-Silva, and E. Soubeiga. Hyperheuristic approaches for multiobjective optimisation. In *Proceedings of the Fifth Metaheuristics International Conference (MIC 2003)*, pages 11.1–11.6, Kyoto, Japan, August 25-28 2003.

[43] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, January 2007.

[44] E. K. Burke, J. L. Silva, A. Silva, and E. Soubeiga. Multi-objective hyper-heuristic approaches for space allocation and timetabling. In *Meta-heuristics: Progress as Real Problem Solvers*, page 129. Springer, 2003.

[45] D. R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley, 1997.

[46] C. L. Li and Z.L. Chen. Binpacking Problem with Concave Costs of Bin Utilization. *Naval Research Logistics*, 53:298–308, 2006.

[47] P. Caamaño, A. Prieto, J. Becerra, F. Bellas, and R. Duro. Real-valued multimodal fitness landscape characterization for evolution. In K. Wong, B. Mendis, and A. Bouzerdoum, editors, *Neural Information Processing. Theory and Algorithms*, volume 6443 of *Lecture Notes in Computer Science*, pages 567–574. Springer Berlin / Heidelberg, 2010.

[48] E. Cantú-Paz. A survey of parallel genetic algorithms. Technical report, IlliGAL 97003. University of Illinois at Urbana-Champaign, 1997.

[49] E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7:311–334, 2001.

[50] M. Caserta and S. Voß. Metaheuristics: Intelligent problem solving. In V. Maniezzo, T. Stützle, S. Voß, R. Sharda, and S. Voß, editors, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 1–38. Springer US, 2010.

[51] K. Chakhlevitch and P. I. Cowling. Hyperheuristics: Recent developments. In C. Cotta, M. Sevaux, and K. Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer, 2008.

[52] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000.

[53] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, 2007.

[54] I. Charon. The noising methods: A generalization of some metaheuristics. *European Journal Of Operational Research*, 135(1):86–101, 2001.

[55] H. Chen, M. Li, and X. Chen. Using diversity as an additional-objective in dynamic multi-objective optimization algorithms. In *Proceedings of the 2009 Second International Symposium on Electronic Commerce and Security - Volume 01*, ISECS '09, pages 484–487, Washington, DC, USA, 2009. IEEE Computer Society.

[56] P.-C. Chen, G. Kendall, and G. Vanden Berghe. An Ant Based Hyper-heuristic for the Travelling Tournament Problem. In *Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched 2007)*, pages 19–26, Honolulu, Hawaii, April 2007.

[57] C. A. Coello and G. B. Lamont, editors. *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore, 2004.

[58] C. A. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer, 2007.

[59] C. A. Coello, G. Toscano, and M. Salazar. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, 2004.

[60] G. Colombo and S. Allen. Problem decomposition for minimum interference frequency assignment. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation, CEC 2007*, pages 3492–3499, 2007.

[61] S. Cook. The P versus NP problem. In *Clay Mathematical Institute; The Millennium Prize Problem*, 2000.

[62] D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, editors. *New ideas in optimization*. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[63] P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 1185–1190, Honolulu, Hawaii, 2002. IEEE Computer Society.

[64] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of 4th Metahuristics International Conference (MIC 2001)*, pages 127–131, Porto Portugal, July 16-20 2001.

[65] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A robust optimisation method applied to nurse scheduling. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, J. L. F.-V. Martín, and H.-P. Schwefel, editors, *PPSN*, volume 2439 of *Lecture Notes in Computer Science*, pages 851–860. Springer, 2002.

[66] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In S. Cagoni, J. Gottlieb, E. Hart, M. Middendorf, and R. Goenther, editors, *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10, Kinsale, Ireland, April 3-4 2002. Springer-Verlag, Springer-Verlag.

[67] P. I. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 176–190, London, UK, UK, 2001. Springer-Verlag.

[68] T. Crainic and M. Toulouse. Parallel Meta-Heuristics. Technical Report CIRRELT-2009-22, May 2009.

[69] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical Report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Universita di Roma "La Sapienza", 1995.

[70] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw-Hill, 2008.

[71] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[72] J. de Armas, G. Miranda, and C. León. Hyperheuristic encoding scheme for multi-objective guillotine cutting problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1683–1690, New York, NY, USA, 2011. ACM.

[73] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, Ann Arbor, MI, USA, 1975.

[74] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* Wiley, Chichester, U.K., 2001.

[75] K. Deb. Evolutionary multi-objective optimization without additional parameters. In Lobo et al. [156], pages 241–257.

[76] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.

[77] K. Deb and M. Goyal. A Combined Genetic Adaptive Search (GeneAS) for Engineering Design. *Computer Science and Informatics*, 26(4):30–45, 1996.

[78] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2002.

[79] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable Test Problems for Evolutionary Multi-Objective Optimization. Technical Report 112, Computer Engineering and Networks Laboratory (TIK), Zurich, Switzerland, 2001.

[80] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[81] T. Dereli and G. S. Das. A hybrid simulated-annealing algorithm for two-dimensional strip packing problem. In *8th International Conference on Adaptive and Natural Computing Algorithms*, volume 4431 of *LNCS*, pages 508–516. Warsaw, Poland, Springer Berlin, April 2007.

[82] J. Dongarra. *High Performance Computing: Technology, Methods and Applications.* Elsevier, New York, 1995.

[83] J. Dongarra et al. *A Users' Guide to PVM Parallel Virtual Machine*, 1991. http://www-unix.mcs.anl.gov/mpi.

[84] K. Dowsland, E. Soubeiga, and E. Burke. A Simulated Annealing Hyper-heuristic for Determining Shipper Sizes. *European Journal of Operational Research*, 179(3):759–774, June 2007.

[85] R. Duncan. A survey of parallel computer architectures. *Computer*, 23:5–16, February 1990.

[86] H. Dyckhoff. A Typology of Cutting and Packing Problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[87] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence*. Morgan Kaufmann (The Morgan Kaufmann Series in Evolutionary Computation), 1 edition, Apr. 2001.

[88] F. Y. Edgeworth. *Mathematical Psychics*. Number edgeworth1881 in History of Economic Thought Books. McMaster University Archive for the History of Economic Thought, 1881.

[89] J. A. Egea, M. Rodríguez-Fernández, J. R. Banga, and R. Martí. Scatter search for chemical and bio-process optimization. *J. of Global Optimization*, 37(3):481–503, March 2007.

[90] A. Eiben and T. Bäck. An empirical investigation of multi-parent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

[91] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124 –141, jul 1999.

[92] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.

[93] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer, Oct. 2008.

[94] H. Esbensen and E. Kuh. Design space exploration using the genetic algorithm. In *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on*, volume 4, pages 500 –503 vol.4, may 1996.

[95] L. Eshelman. The chc adaptive search algorithm. In G. Rawlins, editor, *Foudations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1990.

[96] L. J. Eshelman and D. J. Schaffer. Preventing Premature Convergence in Genetic Algorithms by Preventing Incest. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 115–122. San Francisco, CA: Morgan Kaufmann, 1991.

[97] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67 – 71, 1989.

[98] T. A. Feo, M. G. C. Resende, and S. H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42(5):pp. 860–878, 1994.

[99] M. Fischetti, C. Lepschy, G. Minerva, G. Romanin-Jacur, and E. Toto. Frequency assignment in mobile radio systems using branch-and-cut techniques. *European Journal of Operational Research*, 123(2):241 – 255, 2000.

[100] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheuling Conference*, Carnegie Institute of Technology, 1961.

[101] M. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Trans. Comput.*, C-21:948+, 1972.

[102] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, UK, 1966.

[103] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evol. Comput.*, 3(1):1–16, Mar. 1995.

[104] C. M. Fonseca and P. J. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 584–593, London, UK, UK, 1996. Springer-Verlag.

[105] L. D. Fosdick, E. R. Jessup, G. Domik, and C. J. C. Schauble. *An Introduction to High-performance Scientific Computing*. MIT Press, 1996.

[106] P. Garg. A comparison between memetic algorithm and genetic algorithm for the cryptanalysis of simplified data encryption standard algorithm. *International Journal of Network Security & Its Applications*, 1(1):34 – 42, April 2009.

[107] P. Garrido and C. Castro. Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 255–262, New York, NY, USA, 2009. ACM.

[108] A. Ghosh and S. Dehuri. Evolutionary algorithms for multi-criterion optimization: a survey. *International Journal of Computing & Information Sciences*, 2(1):38–57, April 2004.

[109] C. Glaßer, S. Reith, and H. Vollmer. The complexity of base station positioning in cellular networks. *Discrete Applied Mathematics*, 148(1):1–12, 2005.

[110] F. Glober. Parametric combinations of local job shop scheduling rules. Technical Report 117, Carnegie Mellon University, Pittsburgh, USA, 1963.

[111] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluver Academic Publishers, 2003.

[112] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1998.

[113] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics Journal*, 39:653–684, 2000.

[114] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[115] V. Granville, M. Krivanek, and J.-P. Rasson. Simulated annealing: a proof of convergence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6):652 –656, jun 1994.

[116] J. Gratch and S. Chien. Learning search control knowledge for the deep space network scheduling problem. Technical report, Champaign, IL, USA, 1993.

[117] D. Greiner, J. Emperador, G. Winter, and B. Galván. Improving computational mechanics optimum design using helper objectives: An application in frame bar structures. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, pages 575–589. Springer Berlin / Heidelberg, 2007.

[118] P. Greistorfer. A tabu scatter search metaheuristic for the arc routing problem. *Comput. Ind. Eng.*, 44(2):249–266, Feb. 2003.

[119] W. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497 – 1514, dec. 1980.

[120] L. Han and G. Kendall. Guided operators for a hyper-heuristic genetic algorithm. In T. Gedeon and L. Fung, editors, *AI 2003: Advances in Artificial Intelligence*, volume 2903 of *Lecture Notes in Computer Science*, pages 807–820. Springer Berlin / Heidelberg, 2003.

[121] J. Handl, S. C. Lovell, and J. Knowles. Multiobjectivization by decomposition of scalar cost functions. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, pages 31–40, Berlin, Heidelberg, 2008. Springer-Verlag.

[122] P. B. Hansen. Model programs for computational science: A programming methodology for multicomputers. *Concurrency - Practice and Experience*, pages 407–423, 1993.

[123] L. Hogie. *Mobile Ad Hoc networks: modelling, simulation and broadcast-based applications*. PhD thesis, Le Havre University and Luxembourg University, 2007.

[124] L. Hogie, P. Bouvry, and F. Guinand. An Overview of MANETs Simulation. *Electronics Notes in Theorical Computer Science*, 150(1):81–101, 2006.

[125] L. Hogie, M. Seredynski, F. Guinand, and P. Bouvry. A Bandwidth-Efficient Broadcasting Protocol for Mobile Multi-hop Ad hoc Networks. IEEE, 2006.

[126] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[127] T.-P. Hong, M.-W. Tsai, and T.-K. Liu. Two-dimentional encoding schema and genetic operators. In *JCIS*. Atlantis Press, 2006.

[128] H. Hoos and T. Stützle. *Stochastic local search: foundations and applications*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers, 2005.

[129] J. Horn and et al. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 82–87, 1994.

[130] S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *Evolutionary Computation, IEEE Transactions on*, 10(5):477 –506, oct. 2006.

[131] A. W. Iorio and X. Li. Solving rotated multi-objective optimization problems using differential evolution. In *Proceedings of AI 2004: Advances in Artificial Intelligence*, pages 861–872. LNAI 3339, Springer-Verlag, 2004.

[132] H. Ishibuchi, Y. Hitotsuyanagi, and Y. Nojima. An empirical study on the specification of the local search application probability in multiobjective memetic algorithms. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation, CEC 2007*, pages 2788 –2795, sept. 2007.

[133] H. Ishibuchi and T. Murata. Multi-Objective Genetic Local Search Algorithm. In T. Fukuda and T. Furuhashi, editors, *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 119–124, Nagoya, Japan, 1996. IEEE.

[134] A. Jain and D. Fogel. Case studies in applying fitness distributions in evolutionary algorithms ii. comparing the improvements from crossover and gaussian mutation on simple neural networks. In *Proc. of the 2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 91 –97, 2000.

[135] M. T. Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3:323–347, 2004.

[136] D. E. Joslin and D. P. Clements. "squeaky wheel" optimization. *J. Artif. Int. Res.*, 10(1):353–373, May 1999.

[137] M. Jähne, X. Li, and J. Branke. Evolutionary algorithms and multi-objectivization for the travelling salesman problem. In F. Rothlauf, editor, *Genetic and Evolutionary Computation Conference*, pages 595–602, 2009.

[138] G. Kendall, P. Cowling, and E. Soubeiga. Choice function and random hyperheuristics. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL 2002)*, pages 667–671, Singapore, Nov 2002.

[139] G. Kendall and M. Mohamad. Channel Assignment in Cellular Communication Using a Great Deluge Hyper-Heuristic. In *Proceedings of the 2004 IEEE International Conference on Networks (ICON)*, pages 769–773, Singapore, November 2004.

[140] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942 –1948 vol.4, nov/dec 1995.

[141] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4589):671–680, May 1983.

[142] J. Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. In *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, ISDA '05, pages 552–557, Washington, DC, USA, 2005. IEEE Computer Society.

[143] J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 711–716, Piscataway, NJ, May 2002. IEEE Service Center.

[144] J. Knowles, L. Thiele, and E. Zitzler. A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), Zurich, Switzerland, 2006.

[145] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[146] J. D. Knowles, R. A. Watson, and D. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, EMO '01, pages 269–283, London, UK, 2001. Springer-Verlag.

[147] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Germany, 3rd edition, 2006.

[148] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. A Bradford Book, 1 edition, Dec. 1992.

[149] A. M. J. Kuurne. On GSM mobile measurement based interference matrix generation. In *IEEE 55th Vehicular Technology Conference, VTC Spring 2002*, pages 1965 – 1969, 2002.

[150] H. lan Fang, H. lan Fang, P. Ross, P. Ross, D. Corne, and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 375–382. Morgan Kaufmann, 1993.

[151] C. León, G. Miranda, and C. Rodríguez. *Optimization techniques for solving complex problems*, chapter Divide and Conquer Advanced Techniques, pages 179–192. Wiley, 2008.

[152] C. León, G. Miranda, and C. Segura. METCO: A Parallel Plugin-Based Framework for Multi-Objective Optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–588, 2009.

[153] Z.-f. Z. Li-xiao Ma, Kun-qi Liu and N. Li. Exploring the effects of lamarckian evolution and baldwin effect in differential evolution. In *Communications in Computer and Information Science*, volume 107 of *Computational Intelligence and Intelligent Systems*, pages 127–136. Springer, 2010.

[154] R. F. Linton and T. B. Carroll. *Computational Optimization: New Research Developments*. Nova Science Publishers Inc, 2010.

[155] X. Liu, P. M. Pardalos, S. Rajasekaran, and M. G. C. Resende. A GRASP for Frequency Assignment in Mobile Radio Networks. In B. Badrinath, F. Hsu, P. Pardalos, and S.Rajasekaran, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 52, pages 195–201, 2000.

[156] F. G. Lobo, C. F. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[157] D. Lochtefeld and F. Ciarallo. Multiobjectivization via helper-objectives with the tunable objectives problem. *IEEE Transactions on Evolutionary Computation*, 16(3):373 –390, june 2012.

[158] D. F. Lochtefeld. *Multi-objectivization in Genetic Algorithms*. PhD thesis, 2011.

[159] D. F. Lochtefeld and F. W. Ciarallo. Helper-objective optimization strategies for the job-shop scheduling problem. *Applied Soft Computing*, 11(6):4161 – 4174, 2011.

[160] S. Louis and G. J. E. Rawlins. Pareto optimality, ga-easiness and deception. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 118–123. Morgan Kaufmann, 1993.

[161] H. R. Lourenço, O. C. Marin, and T. Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluver Academic Publishers, 2003.

[162] M. Lozano, D. Molina, and F. Herrera. Editorial Scalability of Evolutionary Algorithms and Other Metaheuristics for Large-scale Continuous Optimization Problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, pages 1–3, 2010.

[163] F. Luna. *Metaheurísticas avanzadas para problemas reales en redes de telecomunicaciones*. PhD thesis, Malga, Spain, Abril 2008.

[164] F. Luna, C. Blum, E. Alba, and A. Nebro. ACO vs EAs for solving a real-world frequency assignment problem in GSM networks. In *Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, pages 94–101, 2007.

[165] F. Luna, C. Estébanez, C. León, J. M. Chaves-González, E. Alba, R. Aler, C. Segura, M. A. Vega-Rodríguez, A. J. Nebro, J. M. Valls, G. Miranda, and J. A. Gómez-Pulido. Metaheuristics for solving a real-world frequency assignment problem in GSM networks. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1579–1586. ACM, 2008.

[166] F. Luna, C. Estébanez, C. León, J. Chaves-González, A. Nebro, R. Aler, C. Segura, M. Vega-Rodríguez, E. Alba, J. Valls, G. Miranda, and J. Gómez-Pulido. Optimization algorithms for large-scale real-world instances of the frequency assignment problem. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 15:975–990, 2011.

[167] F. Luna, C. Estébanez, C. León, J. M. Chaves-González, E. Alba, R. Aler, C. Segura, M. A. Vega-Rodríguez, A. J. Nebro, J. M. Valls, G. Miranda, and J. Gómez-Pulido. Metaheuristics for Solving a Real-World Frequency Assignment Problem in GSM Networks. In *Genetic and Evolutionary Computation Conference*, pages 1579–1586, Atlanta, U.S.A., July 2008. ACM.

[168] A. Lusa and C. N. Potts. A variable neighbourhood search algorithm for the constrained task allocation problem. *The Journal of the Operational Research Society*, 59(6):pp. 812–822, 2008.

[169] J. Macker and M. Corson. Mobile Ad Hoc Networking and the IETF. *ACM Mobile Computing and Communications Review*, 2(1), 1998.

[170] C. Mannino and A. Sassano. An enumerative algorithm for the frequency assignment problem. *Discrete Appl. Math.*, 129(1):155–169, June 2003.

[171] E. Marchiori. Genetic, iterated and multistart local search for the maximum clique problem. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, pages 112–121, London, UK, UK, 2002. Springer-Verlag.

[172] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons Ltd, 1990.

[173] O. Martin and S. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[174] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.

[175] H. Mauch. Closest Substring Problem - Results from an Evolutionary Algorithm. In *11th International Conference on Neural Information Processing (ICONIP)*, volume 3316 of *LNCS*, pages 205–211, Calcutta, India, 2004. Springer.

[176] S. P. Mendes, G. Molina, M. A. Vega-Rodríguez, J. A. Gómez-Pulido, Y. Sáez, G. Miranda, C. Segura, E. Alba, P. Isasi, C. León, and J. M. Sánchez-Pérez. Benchmarking a wide spectrum of metaheuristic techniques for the radio network design problem. *IEEE Transactions on Evolutionary Computation*, 13(5):1133–1150, Oct. 2009.

[177] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, 1994. http://www-unix.mcs.anl.gov/mpi.

[178] H. Meunier, E.-G. Talbi, and P. REININGER. A multiobjective genetic algorithm for radio network optimization. In *In Proceedings of the 2000 Congress on Evolutionary Computation*, pages 317–324. IEEE Press, 2000.

[179] E. Mezura-Montes and A. Palomeque-Ortiz. Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. In E. Mezura-Montes, editor, *Constraint-Handling in Evolutionary Optimization*, volume 198 of *Studies in Computational Intelligence*, pages 95–120. Springer Berlin Heidelberg, 2009.

[180] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, December 2004.

[181] S. Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, pages 7–43, 1996.

[182] G. Miranda. *Algorithmic Skeletons for Combinatorial Optimization - Parallelizations and Applications*. PhD thesis, Tenerife, Spain, May 2009.

[183] M. Mitchell and C. E. Taylor. Evolutionary computation: An overview. *Annual Review of Ecology and Systematics*, 30(1):593–616, 1999.

[184] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997.

[185] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[186] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826, California Institute of Technology, 1989.

[187] M. Mouly and M. B. Paulet. *The GSM System for Mobile Communications*. Mouly et Paulet, Palaiseau, 1992.

[188] J.-B. Mouret. Novelty-based multiobjectivization. In S. Doncieux, N. Bredèche, and J.-B. Mouret, editors, *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, pages 139–154. Springer Berlin / Heidelberg, 2011.

[189] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. A cellular genetic algorithm for multiobjective optimization. In D. A. Pelta and N. Krasnogor, editors, *Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2006)*, pages 25–36, Granada, Spain, 2006.

[190] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. Design issues in a multiobjective cellular genetic algorithm. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Evolutionary Multi-Criterion Optimization. 4th International Conference, EMO 2007*, volume 4403 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2007.

[191] F. Neri, C. Cotta, and P. Moscato. *Handbook of Memetic Algorithms*. Studies in Computational Intelligence. Springer, 2011.

[192] F. Neumann and I. Wegener. Can single-objective optimization profit from multiobjective optimization? In J. Knowles, D. Corne, K. Deb, and D. R. Chair, editors, *Multiobjective Problem Solving from Nature*, Natural Computing Series, pages 115–130. Springer Berlin Heidelberg, 2008.

[193] Q. H. Nguyen, Y. S. Ong, and M. H. Lim. Non-genetic transmission of memes by diffusion. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1017–1024, New York, NY, USA, 2008. ACM.

[194] Q. H. Nguyen, Y. S. Ong, and M. H. Lim. A Probabilistic Memetic Framework. *IEEE Transactions on Evolutionary Computation*, 13(3):604–623, 2009.

[195] S. Nguyen, M. Zhang, M. Johnston, and T. K. Chen. A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In X. Li, editor, *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 3332–3339, Brisbane, Australia, 10-15 June 2012.

[196] J. Nievergelt. Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power. In *Proceedings of the 27th Conference on Current Trends in Theory and Practice of Informatics*, SOFSEM '00, pages 18–35, London, UK, 2000. Springer-Verlag.

[197] M. G. Norman and P. Moscato. A Competitive and Cooperative Approach to Complex Combinatorial Search. Technical Report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena, California, USA, 1989.

[198] OpenMP Architecture Review Board. *OpenMP Application Program Interface. Version 2.5*, 2005. http://www.openmp.org.

[199] P. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, 1997.

[200] V. Pareto. *Cours d'Économie Politique*, volume I and II. Lausanne, 1896.

[201] B. Parhami. *Introduction to parallel processing: algorithms and architectures*. Plenum series in computer science. Plenum Press, 1999.

[202] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[203] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1992.

[204] K. Price. *Dr. Dobb's Journal*, pages 127–132, October 1994.

[205] K. Price, R. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization.* Springer-Verlag, 2006.

[206] P. Rattadilok, A. Gaw, and R. Kwan. Distributed choice function hyper-heuristics for timetabling and scheduling. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 51–67. Springer Berlin / Heidelberg, 2005.

[207] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution.* Frommann-Holzboog, 1973.

[208] C. Ribeiro, I. Rosseti, and R. Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, pages 1–25, 2011.

[209] M. Rocha and J. Neves. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. In I. Imam, Y. Kodratoff, A. El-Dessouki, and M. Ali, editors, *Multiple Approaches to Intelligent Systems*, volume 1611 of *Lecture Notes in Computer Science*, pages 127–136. Springer Berlin Heidelberg, 1999.

[210] P. Ross, J. Marin-Blazquez, and E. Hart. Hyper-heuristics applied to class and exam timetabling problems. In *Congress on Evolutionary Computation, CEC2004*, volume 2, pages 1691 – 1698, june 2004.

[211] Y. Sawaragi, H. Nakayama, and T. Tanino. *Theory of multiobjective optimization.* Academic Press, Orlando, 1985.

[212] J. D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.

[213] H.-P. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation.* John Wiley & Sons, Inc., New York, NY, USA, 1993.

[214] J. Seybold. *Introduction to RF Propagation*. Wiley-Interscience, 2005.

[215] D. Sheskin. *The handbook of parametric and nonparametric statistical procedures*. CRC Press, 2003.

[216] Y. Shi1 and D. Ye. On-line bin packing with arbitrary release times. In *First International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, volume 4614 of *LNCS*, pages 340–349. Hangzhou, China, Springer Berlin, April 2007.

[217] M. K. Simon and M.-S. Alouini. *Digital Communication over Fading Channels: A Unified Approach to Performance Analysis*. Wiley, 2005.

[218] K. Sindhya, A. Sinha, K. Deb, and K. Miettinen. Local search based evolutionary multi-objective optimization algorithm for constrained and unconstrained problems. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 2919–2926, Piscataway, NJ, USA, 2009. IEEE Press.

[219] S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the Eleventh Congress on Evolutionary Computation*, CEC'09, pages 399–406, Piscataway, NJ, USA, 2009. IEEE Press.

[220] J. Smith and T. Fogarty. Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*, number 1141 in Lecture Notes in Computer Science, pages 441–450, 1996.

[221] M. Snir and S. Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA, 1998.

[222] E. Soubeiga. *Development and application of hyperheuristics to personnel scheduling*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, United Kingdom, 2003.

[223] M. Srinivas and L. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656 –667, apr 1994.

[224] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[225] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*. Texts in applied mathematics. Springer, 2002.

[226] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of the 1996 biennial conference of the North American fuzzy information processing society - NAFIPS'96*, pages 519–523. IEEE Press, 1996.

[227] R. Storn and K. Price. Differential Evolution- A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical report, 1995.

[228] P. Strenski and S. Kirkpatrick. Analysis of finite length annealing schedules. *Algorithmica*, 6:346–366, 1991.

[229] G. Sywerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[230] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[231] E.-G. Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009.

[232] E.-G. Talbi and H. Meunier. Hierarchical parallel approach for gsm mobile network design. *J. Parallel Distrib. Comput.*, 66(2):274–290, 2006.

[233] F. Thabtah and P. Cowling. Mining the data from a hyperheuristic approach using associative classification. *Expert Syst. Appl.*, 34(2):1093–1101, Feb. 2008.

[234] A. Toffolo and E. Benini. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11:151–167, May 2003.

[235] R. K. Ursem. Diversity-guided evolutionary algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, PPSN VII, pages 462–474, London, UK, UK, 2002. Springer-Verlag.

[236] Van Der Steen, A. and Dongarra, J. Overview of recent supercomputers - http://www.top500.org/resources/orsc, 2007.

[237] J. A. Vázquez-Rodríguez and S. Petrovic. A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793, Dec. 2010.

[238] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, Jan. 1985.

[239] D. A. V. Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, USA, June 1999.

[240] D. A. V. Veldhuizen, J. B. Zydallis, and G. B. Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 7(2):144–173, 2003.

[241] T. Vinkó and D. Izzo. Learning the best combination of solvers in a distributed global optimization environment. In *Proceedings of Advances in Global Optimization: Methods and Applications (AGO)*, pages 13–17, Mykonos, Greece, June 2007.

[242] K. V. Viswanathan and A. Bagchi. Best-First Search Methods for Constrained Two-Dimensional Cutting Stock Problems. *Operations Research*, 41(4):768–776, 1993.

[243] A. Wai-Sing Loo. *Peer-to-Peer Computing: Building Supercomputers with Web Technologies*. Computer Communications and Networks. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[244] B. H. Walke. *Mobile Radio Networks: Networking, protocols and traffic performance*. Wiley, 2002.

[245] S. Watanabe and K. Sakakibara. A multiobjectivization approach for vehicle routing problems. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, pages 660–672. Springer Berlin / Heidelberg, 2007.

[246] N. Weicker, G. Szabo, K. Weicker, and P. Widmayer. Evolutionary multiobjective optimization for base station transmitter placement with frequency assignment. *Evolutionary Computation, IEEE Transactions on*, 7(2):189–203, April 2003.

[247] T. Weise. *Global Optimization Algorithms - Theory and Application*. 2008.

[248] L. D. Whitley, V. S. Gordon, and K. E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In *Proceedings of the International*

Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, pages 6–15, London, UK, 1994. Springer-Verlag.

[249] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 194–205, 2002.

[250] G. Wilson. *Practical parallel programming*. Scientific and engineering computation. MIT Press, 1995.

[251] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, December 2007.

[252] K. Zielinski and R. Laur. Adaptive parameter setting for a multi-objective particle swarm optimization algorithm. In *The 2007 IEEE Congress on Evolutionary Computation*, pages 3019 –3026, sept. 2007.

[253] K. Zielinski and R. Laur. Stopping criteria for differential evolution in constrained single-objective optimization. In U. Chakraborty, editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 111–138. Springer Berlin / Heidelberg, 2008.

[254] E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. PhD thesis, Zurich, Switzerland, November 1999.

[255] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[256] E. Zitzler and S. Künzli. Indicator-based Selection in Multiobjective Search. In X. Y. et al., editor, *Parallel Problem Solving from Nature - PPSN VIII*, pages 832–842, Birmingham, UK, September 2004. Springer-Verlag. Lecture Notes in Computer Science Vol. 3242.

[257] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimization and Control*, pages 19–26, Barcelona, Spain, 2002. CIMNE.

[258] E. Zitzler and L. Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Computer

Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1998.

[259] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, Apr. 2003.