University of Nottingham

Doctoral Thesis

Hyper-heuristics for Grouping Problems

Author:
Anas Elhag

Supervisor:
Dr. Ender Özcan

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Automated Scheduling, Optimisation and Planning (ASAP) Research Group
School of Computer Science

April 2015

Abstract

Grouping problems are hard to solve combinatorial optimization problems which require partitioning of objects into a minimum number of subsets while another additional objective is simultaneously optimized. Considerable research effort has recently been directed towards automated problem-independent reusable heuristic search methodologies such as hyper-heuristics, which operate on a space formed by a set of low level heuristics rather than solutions, directly. Hyper-heuristics are commonly split into two main categories: selection hyper-heuristics, which are the focus of the work presented in this thesis, and generation hyper-heuristics. Most of the recently proposed selection hyper-heuristics are iterative and make use of two key methods which are employed successively; heuristic selection and move acceptance. At each step, a new solution is produced after a selected heuristic is applied to the solution at hand and then the move acceptance method is used to decide whether the resultant solution replaces the current one or not.

This thesis presents a novel generic single point-based selection hyper-heuristic search framework, referred to as grouping hyper-heuristic framework. The proposed framework deals with one solution at any given decision point during the search process and embeds a fixed set of reusable standard low level heuristics specifically designed for the grouping problems. The use of standard heuristics enables the re-usability of the whole framework across different grouping problem domains with less development effort. The proposed grouping hyper-heuristic framework is based on a bi-objective formulation of any given grouping problem. Inspired from multi-objective optimization, a set of high quality solutions is maintained during the search process, capturing the trade-off between the number of groups and the additional objective for the given grouping problem. Moreover, the grouping framework includes a special two-phased acceptance mechanism that use the traditional move acceptance method only to make a preliminary decision regarding whether to consider the new solution for acceptance or not.

The performance of different selection hyper-heuristics combining different components, implemented based on the proposed framework is investigated on a range of sample grouping problem domains, including graph coloring, exam timetabling and data clustering domains. Additionally, the selection hyper-heuristics performing the best on each domain are compared to the previously proposed problem-specific algorithms from the scientific literature. The empirical results shows that the grouping hyper-heuristics built based on the proposed framework are not only sufficiently general, but also able to obtain high quality solutions, competitive to some previously proposed approaches. The selection hyper-heuristic employing the 'reinforcement learning' heuristic selection method and embedding the 'iteration limited threshold accepting' move acceptance method performs the best in the overall across those grouping problem domains.

Acknowledgements

All praise is due to the Almighty God for the countless blessings He showered upon me throughout my journey in life, and for all the physical and mental strength He bestowed on me during the activities of preparing this thesis and conducting the associated research.

First and foremost, my deepest gratitude and thanks go to my principal supervisor Dr. Ender Özcan, who guided me through every single step along the way for the last few years, and helped me become the person who I am today. Without this immense support, advice and above all the unparalleled patience of Dr Özcan, this thesis would not have seen the light.

I would also like to give a special heartfelt mention to the Lancaster, Nottingham, Cardiff and Southampton (LANCS) initiative as well as all the members of the Automated Scheduling, optimisAtion and Planning (ASAP) group for their unlimited support, academically, administratively and financially, throughout my journey of writing this thesis, as well as for all the other huge efforts they make to ensure that all the PhD students always have the best environment and the most accessible sources of advice and support.

Writing this doctoral thesis would not have been possible without the unlimited support and help of a very long list of kind people around me, including - but not limited to - family members, friends, academic and administrative staff members of the School of Computer Science and Information Technology as well as my colleagues in the Officers Team and the staff members of the Students' Union at the University of Nottingham.

Last but not least, I would like to thank my parents, who always supported me in every way possible, and did every thing they could to ensure that my progress, health and welfare are always in the best state, and always prayed for me to have a successful life and career. There is no way I will ever be able to pay you back. I can only promise you that I will do my best to be the son you want me to be, and the person who is always willing to do whatever it takes to put a smile on your lovely faces.

Contents

List of Figures

List of Tables

List of Abbreviations

ADHS	**A**daptive **D**ynamic **H**euristic **S**et
ALL	Accepts **ALL**
BFD	**B**est **F**it **D**escent
CF	**C**hoice **F**unction
CHeSC	**C**ross-Domain **H**euristic **S**earch **C**hallenge
EF	**E**xact **F**it
FF	**F**irst **F**it
FFD	**F**irst **F**it **D**escent
GA	**G**enetic **A**lgorithm
GDEL	**G**reat **Del**uge
GE	**G**roup **E**ncoding
GGAE	**G**rouping **G**enetic **A**lgorithm **E**ncoding
GP	**G**enetic **P**rogramming
IEQ	**I**mproving or **Eq**ual
ILTA	**I**terated **L**imited **T**hreshold **A**ccepting
LACC	**L**ate **Acc**eptance
LDD	**L**argest **D**egree **D**escent
LF	**L**argest **F**it
LLE	**L**inear **L**inkage **E**ncoding
LLE-b	**L**inear **L**inkage **E**ncoding with **B**ackward Links
LLE-e	**L**inear **L**inkage **E**ncoding with **E**nding Node Links
LLH	**L**ow **L**evel **H**euristic
MA	**M**emetic **A**lgorithm
MISTA	**M**ultidisciplinary **I**nternational Conference on **S**cheduling: **T**heory and **A**pplications

MPCF	**M**achine-**P**art **C**ell **F**ormation
NE	**N**umeric **E**ncoding
OI	**O**nly **I**mproving
RD	**R**andom **D**escent
RL	**R**einforcement **L**earning
RP	**R**andom **P**ermutation
SA	**S**imulated **A**nnealing
SCOR	**S**tudent **C**onference on **O**perational **R**esearch
SDD	**S**aturation **D**egree **D**escent
SR	**S**imple **R**andom
SSE	**S**quared **S**um of **E**rrors
TS	**T**abu **S**earch
UKCI	**U**nited **K**ingdom Workshop on **C**omputational **I**ntelligence

To my parents

Chapter 1

Introduction

1.1 Background and Motivation

In many cases, the search space size becomes so immense, the use of an exact method exhaustively searching that space for an optimization problem is not viable. Alternatively, (guided) search methodologies known as heuristics and meta-heuristics which prioritize the search speed over the guarantee of finding optimal solutions have been applied to such problems with varying degrees of success. However, the resulting algorithms are often not reusable and require expert intervention before they can be applied for different problems or even different instances of the same problem [Burke et al., 2003].

Hyper-heuristics are emerging high-level heuristic search techniques that attempt to solve such complex optimization problems by automating the process of exploring search spaces of problem-specific heuristics (or components of such heuristics). One of the main aims of hyper-heuristics is to raise the level of generality at which the optimization systems operate. They attempt to produce optimization systems that are capable of finding "good-enough, soon-enough, cheap-enough" solutions to as many optimization problems as possible [Burke et al., 2003, Ross, 2005]. There are two main types of such search methodologies: selection and generation hyper-heuristics. A selection hyper-heuristic [Burke et al., 2009b] explores the search space of heuristics by *selecting* a heuristic from a given set of problem specific heuristics (and/or search operators) and apply it on the current solution(s) at each decision point. They operate at a high level of abstraction in order to, hopefully, achieve good results when provided with different sets of low level heuristics and applied on different problem domains. In the general hyper-heuristic framework shown in Figure 1.1 (a), it is assumed that there is a domain barrier that separates the hyper-heuristic strategy from the given problem domain. Consequently, selection hyper-heuristics have no knowledge about the problem

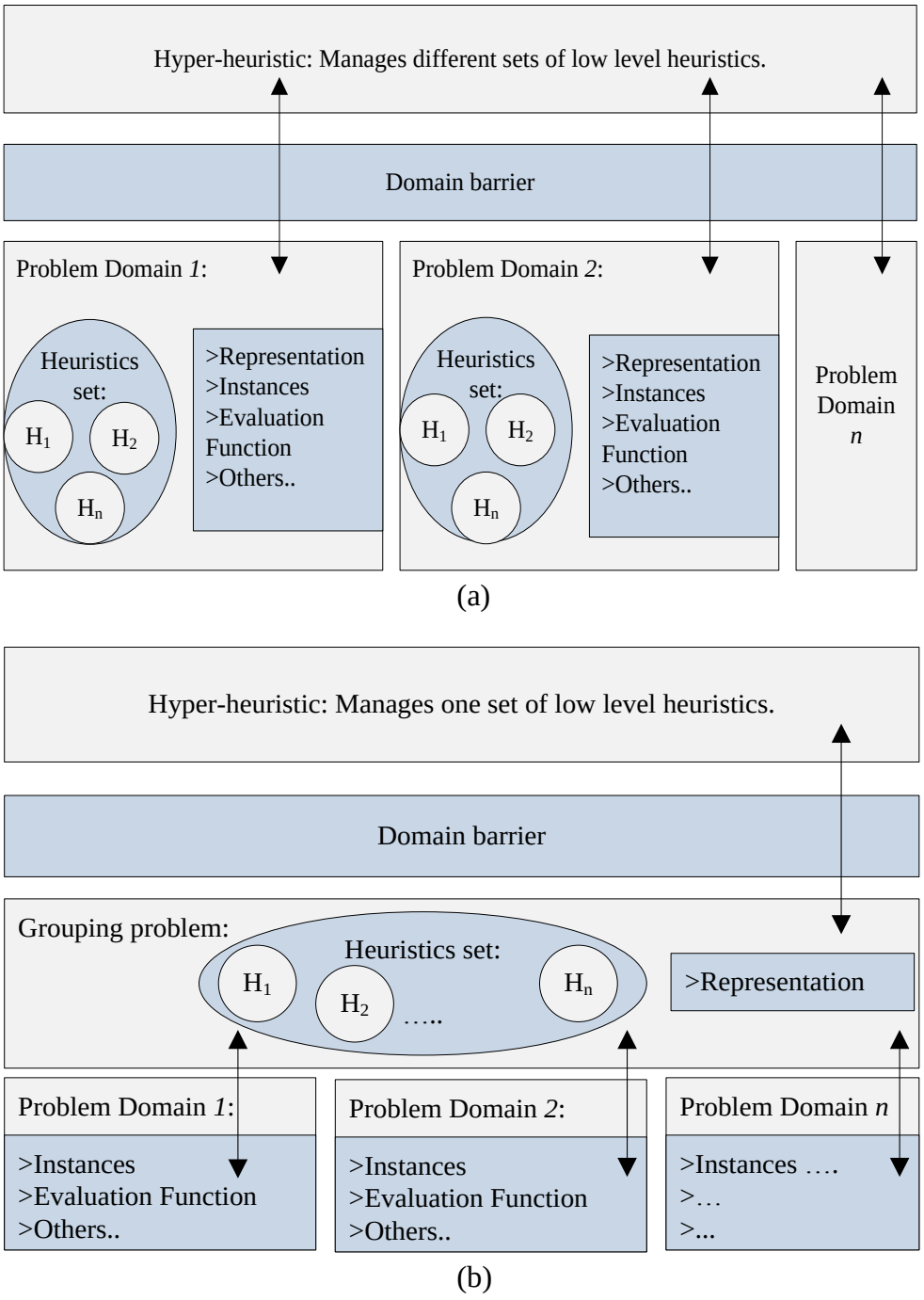

FIGURE 1.1: A comparison between the conceptual layouts of a generic hyper-heuristic framework and the proposed grouping hyper-heuristic framework. (a)A generic hyper-heuristic framework: a different set of low level heuristics for each domain, and (b)The proposed grouping framework: a fixed set of low level heuristics for all grouping problems.

domains and their implementation, or even the nature of the low level heuristics. They are only allowed to call upon the given sets of low level heuristics and monitor their

performance and the quality of solutions they return. The research presented in this thesis is concerned with the application of selection hyper-heuristics on an important class of combinatorial optimization problems known as Grouping Problems.

The task in a grouping problem is to partition a large group, U, of n items, $U = x_1, x_2, x_3, ..., x_n$, into a collection of k $(2 \leq k \leq n-1)$ subgroups, u_i $(1 \leq i \leq k)$, such that each item appears exactly in one subgroup. Such grouping (partitioning) problem arises in many real world and industrial problems such as packing and timetabling, most of which are known to be NP-hard [Falkenauer, 1998]. In general, not all groupings are feasible for a given problem, since different grouping problems impose different constraints and so they introduce different objectives. For example, in examination timetabling, the examinations of a student must not be in the same group indicating that they will take place at the same time, while in data clustering, two dissimilar items must not be placed into the same group. Yet, a common feature of such grouping problems is that they require the number of groups (k) to be minimized as well. Moreover, the overall quality/fitness (f) of a solution to a grouping problem $(U_g = \{u_1, u_2, u_3, ..., u_k\})$ can be measured using a decomposable evaluation function which can accumulate the partial contribution from each group as in Eq. (1.1) below.

$$f(U_g) = \sum_{i=1}^{k} f(u_i) \tag{1.1}$$

Solving a grouping problem using an exact algorithm might not be attainable, requiring alternative solution methods such as heuristics. Often, each grouping problem is tackled individually, and many problem specific solution methodologies for a given problem are available in the literature. These methodologies have either been developed based on different solution representations or specifically tailored to fit each particular problem domain. As a result, in almost all cases, these problem specific methodologies have to be heavily modified in order to be applicable on other grouping problems, or even, sometimes, other instances of the same problem. Grouping problems such as data clustering, exam timetabling and graph coloring can be formulated as problems with multiple objectives. More precisely, they can be formulated as bi-objective problems, in the respect that the minimum number of clusters causing the least error, the minimum number of time slots causing the least number of violations and the minimum number of colors causing the least conflicts are sought, respectively.

There are different frameworks for implementing hyper-heuristics, such as HyFlex [Burke et al., 2009a] and Hyperion [Brownlee et al., 2014, Swan et al., 2011]. A comprehensive overview of hyper-heuristic frameworks is provided in Ryser-Welch and Miller [2014]. Among those frameworks, HyFlex is the most widely known interface. HyFlex provides

a common interface which defines access methods for the high level control algorithm operating as a hyper-heuristic to the low level domain. This way, the details of the problem domain implementation, such as the data structures for representing candidate solutions, objective function, implementation of the low level heuristics are kept hidden from the hyper-heuristic implementer yielding a simple interface. HyFlex v1.0 is the Java implementation of this interface supporting the design of cross-domain heuristic search methods [Burke et al., 2009a] embodying implementation of six different problem domains. Hence, the hyper-heuristics developers can focus on the design of the high-level strategy that can control and configure the algorithmic components, and solve a given instance from a given domain. First Cross-Domain Heuristic Search Challenge (CHeSC 2011) [Burke et al., 2011] was organized recently using HyFlex v1.0 to determine the state-of-the-art selection hyper-heuristic. CHeSC 2011 eventually became a benchmarking tool for performance comparison of selection hyper-heuristics across different problem domains. None of the previous domain implementations based on the previously proposed frameworks contain a general grouping problem domain implementation.

1.2 Aims and Scope

The main aim of this research is to provide a generic single point-based search selection hyper-heuristic framework in which low level heuristics are fixed for solving different grouping problems. In HyFlex v1.0, each problem domain has a separate implementation with a different set of low level heuristics. However, the aim of this research is to raise the level of generality of the implementation of a problem domain more and provide a generic framework embedding a fixed set of low level heuristics for solving a given grouping problem. This means that an implementer does not need to worry about the low level heuristic implementation and other data structures required during the search process as shown in Figure 1.1 (*b*). However, domain expert still needs to deal with the implementation of the other components, such as the objective function, instance reader.

In order to achieve the aims of this study, the following objectives were outlined:

- To investigate different grouping problems and identify their common objectives and features.
- To study existing grouping solution representations and heuristics/search operators applied for grouping problems and identify their strengths and weaknesses.

- To understand existing meta-heuristic and hyper-heuristic methodologies, with particular focus on selection hyper-heuristics.
- To study existing multi-objective optimization techniques, and identify main concepts that can be adopted for grouping problems.
- To design a single point based search selection hyper-heuristic framework that can be applied to all grouping problems.
- To develop a set of generic search operators that can be applied to all grouping problems without modification. This set should be designed such that it can efficiently produce high quality solutions on benchmark and real-world problems.

In this thesis, a selection hyper-heuristic framework embedding a fixed set of low level grouping heuristics is presented. Nine different selection hyper-heuristics combining the simple random, adaptive dynamic heuristic set [Misir et al., 2013] (component of the hyper-heuristic which won the CHeSC 2011 competition) and reinforcement learning [Nareyek, 2004] heuristic selection methods with the late acceptance [Burke and Bykov, 2008], great deluge Dueck [1993] and the iteration limited threshold accepting [Misir et al., 2013] (component of the hyper-heuristic which won the CHeSC 2011 competition) move acceptance methods are used to evaluate the framework on problem instances from the graph coloring, exam timetabling, and data clustering problem domains. The benchmark instances are taken from the DIMACS challenge suite[1], the Toronto benchmark [Qu et al., 2009], the UCI Machine Learning Repository [Bache and Lichman, 2013] and hand-crafted clustering instances of Handl and Knowles [2007].

The empirical results show that a learning selection hyper-heuristic developed using the framework turns out to be indeed sufficiently general and reusable. This hyper-heuristic either beats most of the previously proposed approaches tailored for the specific problems in hand or shows that it is highly competitive to those approaches.

1.3 Contributions of the Thesis

1.3.1 Summary of Contributions

In this study, we describe a novel selection hyper-heuristic framework for grouping problems. The proposed framework is based on a bi-objective formulation of any given grouping problem. This contributes to the considerable research effort that has recently been directed towards the automated, problem-independent search methodologies known as

[1] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

hyper-heuristics. Additionally, the proposed framework embeds a fixed set of reusable standard low level heuristics that are specifically designed for grouping problems. The introduction of this fixed set of low level heuristics in the grouping framework contributes to changing the traditional hyper-heuristic paradigm as shown in Figure 1.1 (a) to (b). Also, a special two-phased acceptance mechanism is presented. This acceptance mechanism changes the way traditional move acceptance methods operate since since it uses the traditional move acceptance only to make a preliminary decision about whether to consider the new solutions for acceptance or not.

1.3.2 Academic Publications Produced

The following publications were produced as a direct result of the work undertaken during the course of conducting this research:

- Anas Elhag and Ender Özcan, Grouping hyper-heuristic framework: application on Data Clustering, *in preparation*. [journal]
- Anas Elhag and Ender Özcan, A Grouping Hyper-Heuristic Framework: Application on Graph Colouring, Expert Systems With Applications, DOI: 10.1016/j.eswa.2015.01.038, available online from 2015. [journal]
- Anas Elhag and Ender Özcan (2013), A grouping hyper-heuristic framework based on linear linkage encoding for graph coloring, The 13th annual workshop on computational intelligence (UKCI), pp. 321-326. [conference]
- Anas Elhag and Ender Özcan (2013), A Grouping hyper-heuristic for graph coloring, The 6th multidisciplinary international conference on scheduling: theory and applications (MISTA), pp. 770-773. [extended abstract]

An abstract entitled "A hybrid encoding scheme for grouping problems" was also presented at the 3rd student conference on operational research (SCOR), 2012.

1.4 Thesis Structure

1.4.1 Chapter 1: Introduction

This chapter introduces the thesis topic and relevant concepts. It also provides a list of the aims of the thesis, as well as the scientific contributions and papers published during the study.

1.4.2 Chapter 2: Heuristic Search Methodologies

In this chapter, a literature survey of hyper-heuristics is presented, highlighting the recent developments in selection hyper-heuristics and cross domain search methodologies. Also, a review of the main concepts borrowed from the multi-objective optimization domain is provided.

1.4.3 Chapter 3: Grouping Problem

This chapter provides a literature survey of grouping problems, with a critical discussion about the main grouping solution representations and associated search operators used in the scientific literature. The chapter also provides a survey of selected grouping problem domains, covering the scientific literature studies that are used in the reminder of the dissertation for comparison purposes.

1.4.4 Chapter 4: Single Point-Based Selection Hyper-heuristics for Grouping Problems

This chapter presents a detailed description of the proposed single point-based selection grouping hyper-heuristic framework, explaining all the relevant algorithmic specifications with a focus on a proposed two-phased solution acceptance mechanism.

1.4.5 Chapter 5: Preliminary Analysis of Grouping Hyper-heuristics' Components

This chapter provides a comparison between three implementations of the grouping framework presented in the previous chapter, using three different solution representations from the scientific literature. The chapter also provides a comparison between the performance of the grouping framework and the performance of a traditional hyper-heuristic approach.

1.4.6 Chapter 6: Application of Grouping Hyper-heuristics on Graph Coloring, Timetabling and Data Clustering

In this chapter, the performance of the framework using nine different selection hyper-heuristics on a set of well known graph coloring, exam timetabling and data clustering problem instances is presented and analyzed.

1.4.7 Chapter 7: Conclusions

This chapter presents a summary of the work undertaken in this research along with recommendations for future work.

Chapter 2

Heuristic Search Methods

In the context of optimization and operations research, many real world problems are proven to be computationally intractable, mainly because of their huge search space. For such problems, it is impractical to try to find either optimal, or even good quality, solutions by means of exhaustive search. Instead, researchers often use guided search methodologies known as *heuristics* which are incomplete search methods, most of which involve random components. The main feature of these heuristics is that they sacrifice the guarantee of solution optimality for the sake of the search speed; yet, some of these techniques can be relatively slow. An important drawback of these methods is that they represent bespoke problem-specific methods; i.e. they are designed and tuned for a specific real world problem. In addition to the high cost associated with the development and implementation of such bespoke methods, the resulting algorithms are often not reusable, and they produce low-quality solutions when applied on different problems, or in some cases, different instances of the same problem [Burke et al., 2013, 2003].

Different heuristics have the same average performance when applied over all problems defined on a given finite search space [Burke et al., 2003]. In other words, each heuristic has some strengths and weaknesses, and no single heuristic has absolute superiority over another, on average. As a result of this, a new research direction that tries to study the effects of *combining* different heuristics (or heuristic components) in new algorithms to solve the given problems, instead of using a single heuristic on its own has emerged [Ross, 2005]. The underlying assumption is that combining known heuristics (or heuristic components) may allow them to compensate for the weaknesses of each other when applied on different problems or different instances of the same problem. Although the term "hyper-heuristic" is not new and the idea could be traced back to the early 1960s, the term was independently used in [Cowling et al., 2000] in 2000 in the context of automated theorem proving to refer to a protocol that combines several artificial

intelligence methods [Burke et al., 2009b]. In contrast to a traditional heuristic which directly explores a space of problem solutions, a hyper-heuristic can be defined as a heuristic that explores a space of heuristics [Burke et al., 2003, Ross, 2005]. Basically, the key idea is to "choose" a heuristic from a set of simple and well known problem specific heuristics, to transform the problem at each decision point. For example, if we consider the bin packing problem, a good combination of the "Largest Fit (LF)", "First Fit (FF)" and the "Exact Fit (EF)" procedures might outperform either of them alone [Burke et al., 2003, Ross, 2005]. However, this basic definition has been slightly modified as will be shown in later in this chapter.

Hyper-heuristics aim at raising the level of generality at which optimization systems can operate. The ultimate research goal is to produce a hyper-heuristic that is applicable across all problem domains in order to address the needs of organizations and users who are interested in "good-enough, soon-enough, cheap-enough" solutions to their optimization problems [Burke et al., 2003, Ross, 2005]. In many real-world problem domains, there are broad sectors of users who need "good-enough, soon-enough, cheap-enough" solutions to their search problems. A high school with a timetabling problem or a small delivery company with a routing problem would prefer to use such a system to get a reasonable-quality solution rather than to pay the cost of expensive manually-parameter-tuned heuristics. For this kind of users, " *"good-enough" often means solutions better than they currently obtain by hand, "soon-enough" typically means solutions delivered at least as quick as those obtained by hand, and "cheap-enough" usually means the cost of the system is low enough that its solutions add value to the organization"* " [Hyde, 2010]. Clearly, there is a trade-off between the generality of a hyper-heuristic, and the quality of the solutions it obtains [Burke et al., 2003, Ross, 2005].

2.1 Meta-heuristics

By mid 1980s, Glover [1986] introduced the term *meta-heuristic* to describe tabu search technique. Recently, meta-heuristic is defined as a high-level strategy that is designed to guide the search process for any given problem [Sörensen and Glover, 2013].

Table 2.1 provides a summary of some applications used meta-heuristics as search methodologies.

The meta-heuristic technique can be divided into two main categories: *single-point* techniques that deal with a single solution at each iteration (e.g. tabu search), and *population-based* techniques which employ a population of solutions (e.g. evolutionary

TABLE 2.1: Applications used meta-heuristics as search methodologies

Application	Meta-heuristic Method [Reference]
Routing-packing problem	Local search [Ceschia et al., 2013]
Vehicle routing problem	Tabu search [Ceschia et al., 2011]
Job shop scheduling problem	Tabu search [Hurink and Knust, 2005]
School timetabling problem	Simulated annealing [Abramson et al., 1999]
Traveling salesman problem	Great deluge [Dueck, 1993]
Examination timetabling problem	Late acceptance [Özcan et al., 2009]
Course timetabling problem	Late acceptance [Abuhamdah, 2010]
Nurse rostering problem	Memetic algorithm [Özcan, 2007]
Generalized traveling salesman	Memetic algorithm [Gutin and Karapetyan, 2009]

algorithms). In the following subsections, some of the most popular meta-heuristic techniques are described.

2.1.1 Local Search

Local search algorithm [Ceschia et al., 2013] starts with an initial solution, and iteratively moves from a solution to another neighbor solution through some modification.

Hill climbing is a special type of local search which moves to a neighbor solution only if its quality is better than the candidate solution.

Traditionally, a hill climbing is liable to be trapped in a local optimum solution at which the qualities of all neighboring solutions are worse. Algorithm 1 provides the pseudocode of hill climbing.

Algorithm 1 Hill climbing local search

1: $S \leftarrow S_{initial}$; {construct an initial solution}
2: $S_{best} \leftarrow S$;
3: $f \leftarrow Evaluate(S)$; {calculate the initial objective function value}
4: **repeat**
5: $S' \leftarrow GenerateNeighbour(S)$; {generate a neighbor solution}
6: $f' \leftarrow Evaluate(S')$; {calculate the objective function value of the new solution}
7: **if** f' isBetterThan f **then**
8: $S_{best} \leftarrow MaintainBestSolution(S')$; {the new solution S' is maintained as the best solution S_{best} found so far}
9: $S \leftarrow S'$;
10: $f \leftarrow f'$;
11: **end if**
12: **until** (termination criteria are satisfied);
13: **return** S_{best};

2.1.2 Simulated Annealing (SA)

Simulated annealing [Abramson et al., 1999] is a stochastic-based method, designed to escape from the local optima. The technique is inspired by the process of annealing in metallurgy.

SA accepts the non-improving moves with a probability defined as:

$$probability = e^{-\frac{\Delta}{T}} \tag{2.1}$$

where Δ is the change in the fitness, and T is a parameter called *temperature*.

In the SA algorithm, the search process starts at a high temperature giving more chance to accept non-improving moves; the value is gradually reduced as the system cools making the probability of accepting non-improving moves at a much lower rate. The pseudocode of simulated annealing method is shown in Algorithm 2.

Algorithm 2 Simulated annealing

1: $S \leftarrow S_{initial}$; {construct an initial solution}
2: $S_{best} \leftarrow S$;
3: $f \leftarrow Evaluate(S)$; {calculate the initial objective function value}
4: **repeat**
5: $Set(T)$; {set/update the temperature parameter}
6: $S' \leftarrow GenerateNeighbour(S)$; {generate a neighbor solution}
7: $f' \leftarrow Evaluate(S')$; {calculate the objective function value of the new solution}
8: **if** (f' isBetterThan f) **or** ($e^{-\frac{\Delta}{T}} <$ RandomBetween[0,1]) **then**
9: $S_{best} \leftarrow MaintainBestSolution(S')$; {the new solution S' is maintained as the best solution S_{best} found so far}
10: $S \leftarrow S'$;
11: $f \leftarrow f'$;
12: **end if**
13: **until** (termination criteria are satisfied);
14: **return** S_{best};

2.1.3 Great Deluge (GDEL)

Great deluge is introduced by Dueck [1993], inspired from the analogy that a person climbing a hill in a great deluge, will intuitively attempt to move to a direction so that the feet does not get wet hoping to find a higher ground as the water level rises.

Similar to SA, GDEL accepts the non-improving moves. However, the non-improving move is only accepted if its quality is better than a *water level*. The basic outline of great deluge method is provided in Algorithm 3.

Algorithm 3 Great deluge

1: $S \leftarrow S_{initial}$; {construct an initial solution}
2: $S_{best} \leftarrow S$;
3: $f \leftarrow Evaluate(S)$; {calculate the initial objective function value}
4: **repeat**
5: $Set(WaterLevel)$; {set/update the water level}
6: $S' \leftarrow GenerateNeighbour(S)$; {generate a neighbor solution}
7: $f' \leftarrow Evaluate(S')$; {calculate the objective function value of the new solution}
8: **if** (f' isBetterThan f) **or** (f' isBetterThan $WaterLevel$) **then**
9: $S_{best} \leftarrow MaintainBestSolution(S')$; {the new solution S' is maintained as the best solution S_{best} found so far}
10: $S \leftarrow S'$;
11: $f \leftarrow f'$;
12: **end if**
13: **until** (termination criteria are satisfied);
14: **return** S_{best};

2.1.4 Tabu Search (TS)

Tabu search was developed by Glover [1986]. Tabu search maintains a list of visited areas of the search space, and disallows visiting these areas of the search space. This could potentially lead the search process to explore more solution space and hopefully avoid the local optima. The basic outline of tabu search method is provided in Algorithm 4

Algorithm 4 Tabu search

1: $S \leftarrow S_{initial}$; {construct an initial solution}
2: $S_{best} \leftarrow S$;
3: $f \leftarrow Evaluate(S)$; {calculate the initial objective function value}
4: **repeat**
5: $S' \leftarrow GenerateNeighbour(S)$; {generate a neighbor solution not in the list}
6: $AddToTabuList(S')$;
7: $f' \leftarrow Evaluate(S')$; {calculate the objective function value of the new solution}
8: **if** f' isBetterThan f **then**
9: $S_{best} \leftarrow MaintainBestSolution(S')$; {the new solution S' is maintained as the best solution S_{best} found so far}
10: $S \leftarrow S'$;
11: $f \leftarrow f'$;
12: **end if**
13: **until** (termination criteria are satisfied);
14: **return** S_{best};

2.1.5 Late Acceptance Hill Climbing (LACC)

Late acceptance hill climbing [Burke and Bykov, 2008] compares the quality of the neighbour solution to a solution generated a number of steps before, as opposed to the candidate solution.

LACC maintains a queue list of n recently generated solutions, and at each step, a candidate solution is accepted if its quality is better than the quality of the solution stored at location n in the queue list. Algorithm 5 provides the pseudocode of the LACC approach.

Algorithm 5 Late acceptance hill climbing

$S \leftarrow S_{initial}$; {construct an initial solution}
$S_{best} \leftarrow S$;
$f \leftarrow Evaluate(S)$; {calculate the initial objective function value}
$Initialise(list)$; {initialize the queue list of size n}
repeat
  $S' \leftarrow GenerateNeighbour(S)$; {generate a neighbor solution}
  $f' \leftarrow Evaluate(S')$; {calculate the objective function value of the new solution}
  if f' isBetterThan quality of solution at location n in the list **then**
    $S_{best} \leftarrow MaintainBestSolution(S')$; {the new solution S' is maintained as the best solution S_{best} found so far}
    $S \leftarrow S'$;
  end if
  $Update(list)$; {update the queue list of size n}
until (termination criteria are satisfied);
return S_{best};

2.1.6 Genetic Algorithm (GA)

Genetic algorithm is a population-based meta-heuristic, inspired by the principles of evolution in nature [Sastry et al., 2014]. A solution is represented by a chromosome. The algorithm iteratively updates a pool of solutions and produces new generation of solutions through the use of operators which alter a solution (mutation) and combine solutions (crossover). A simple genetic algorithm is provided in Algorithm 6

Algorithm 6 Genetic algorithm

$GeneratePopulationOfSolutions$;
repeat
  $EvaluateSolutionsInPopulation$;
  $SelectSolutionsFromPopulation$;
  $ApplyCrossoverWithProbability$;
  $ApplyMutationWithProbability$;
until (termination criteria are satisfied);
return $BestSolutionObtained$;

2.1.7 Memetic Algorithm (MA)

Memetic algorithm is a hybridization of genetic algorithm and local search methods [Moscato, 1989]. The introduced local search method is commonly applied after crossover and mutation have been performed, as shown in Algorithm 7.

Algorithm 7 Memetic algorithm

```
1: GeneratePopulationOfSolutions;
2: repeat
3:    EvaluateSolutionsInPopulation;
4:    SelectSolutionsFromPopulation;
5:    ApplyCrossoverWithProbability;
6:    ApplyMutationWithProbability;
7:    ApplyLocalSearch;
8: until (termination criteria are satisfied);
9: return BestSolutionObtained;
```

2.2 Hyper-heuristics

A hyper-heuristic operates on a search space of pre-defined low level heuristics (LLHs), and ideally at a very high level of abstraction such that it has no knowledge about the problem domain or about the implementation or even the nature of the LLHs themselves. It is only allowed to interact with the LLHs, call upon them and monitor their performance and the quality of solutions they return. This abstraction of the hyper-heuristic is required in order to allow the same hyper-heuristic to, hopefully, achieve good results when provided with different sets of LLHs and applied on different problem domains. The domain barrier prevents the hyper heuristic strategy from accessing any domain specific information [Burke et al., 2003, Ross, 2005].

Researchers have proposed a variety of hyper-heuristics that use different high level strategies and applied them to different combinatorial problem domains. Some researchers tried to propose unified classifications for different hyper-heuristic approaches. The following subsections gives an overview of the major hyper-heuristics classifications.

2.2.1 Three Preliminary Classifications

The first classification of hyper-heuristics was proposed in [Soubeiga, 2003] in 2003, which categorizes hyper-heuristics into two types:

1. Without learning, and

2. With learning.

In the category of hyper-heuristics without learning, a set of heuristics or neighborhood structures are available, and one of them is used at each decision point. The main feature of this category is that no learning mechanism is provided; and the available heuristics are applied either in a certain pre-determined order as in [Dowsland, 1998] or chosen in a more flexible fashion depending on the qualities of the current solution as in [Gamboa et al., 2006].

In the second category, methods that dynamically learn and modify the preferences of the LLHs based on their historical performance are used to choose between several LLHs. The author further distinguishes between the hyper-heuristics that use genetic algorithms (GAs) for learning [lan Fang et al., 1994] and those which do not [Nareyek, 2003].

Two years later, a second classification was proposed in [Bai, 2005] where hyper-heuristics are classified into those which are

1. Constructive, and those which are
2. Local search methods.

Constructive hyper-heuristics build a solution gradually by choosing from a set of constructive LLHs. In other words, a constructive hyper-heuristic searches for a good sequence of heuristics which can construct a good solution. Usually, the LLHs are well-known constructive heuristics such as the *Best-Fit Descent (BFD)* and the *First-Fit Descent (FFD)* for Bin packing problems [Hyde, 2010], *Saturation Degree Descent (SDD)* and *Largest Degree Descent (LDD)* [Carter, 1986] for exam timetabling problems. Each of these simple heuristics can create an efficient solution when applied individually, but can also create a poor quality solution if they get trapped into local optima. Constructive hyper-heuristics attempt to handle this situation by synchronizing these simple heuristics and chose the most appropriate one at each decision point.

In the literature, most of the proposed constructive hyper-heuristics are GA based approaches such as in [Hart et al., 1998]. Other meta-heuristic approaches have recently been employed as high-level strategies in constructive hyper-heuristic frameworks such as in [Burke et al., 2007c].

On the other hand, the local search hyper-heuristics start from a complete initial solution and iteratively select, from a set of neighborhood structures, the appropriate heuristics to lead the search in a promising direction. At each decision point, a local search hyper-heuristic tries to find the “right” heuristic to guide the search in a promising direction

and improve the best solution found so far. Some hyper-heuristics which belong to this category make use of what is known as the "*Choice Function (CF)*" which dynamically selects suitable heuristics at each decision point. The Choice Function based hyper-heuristics were found to be superior to applying the heuristics randomly [Bai, 2005].

A third classification of hyper-heuristics was proposed when some researchers started using genetic programming (GP) for hyper-heuristic which led to the emergence of a new class of hyper-heuristics as well as to the modification of the definition of hyper-heuristics.

Before the emergence of this new class, researchers only considered selection hyper-heuristics, in which the hyper-heuristics framework was provided with a set of well-known heuristics. In contrast, GP hyper-heuristics introduced the idea of generating completely new heuristics from the basic components of the provided heuristics, by firstly decomposing them into their basic building blocks and then recombining them in a new way [Burke et al., 2009b]. Examples of this approach can be found in [Bader-El-Den and Poli, 2007, Burke et al., 2006a, 2007a]. After this work, hyper-heuristics have naturally been classified into "selection hyper-heuristics" and "generation hyper-heuristics".

2.2.2 Selection Hyper-heuristics

As discussed in [Özcan et al., 2008], selection hyper-heuristics perform two main steps:

1. Heuristic selection, and
2. Move acceptance.

Within the constraints outlined above, there are - incredibly - so many different ways for achieving each one of these steps. Selection hyper-heuristic is the focus of this work. Algorithm 8 provides the pseudocode of the generic selection hyper-heuristic framework.

2.2.2.1 Heuristics Selection Methods

So many different methods are being used to select one of the LLHs at any decision point. For instance, in some hyper-heuristics, the selection of a LLH is done absolutely randomly, which is the oldest and the simplest way in the literature. There are many variations of this approach such as the *Simple Random (SR)* which chooses a LLH randomly based on a uniform probability selection [Bai and Kendall, 2005, Burke et al., 2005, Cowling and Chakhlevitch, 2003], the *Random Permutation (RP)* which begins

Algorithm 8 Generic selection hyper-heuristic framework

```
1: S ← S_initial; {construct an initial solution}
2: S_best ← S;
3: f ← Evaluate(S); {calculate the initial objective function value}
4: repeat
5:    LLH_i ← Select(LLH); {select low level heuristic LLH_i}
6:    S' ← Apply(LLH_i, S); {apply LLH_i to the candidate solution}
7:    f' ← Evaluate(S'); {calculate the objective function value of the new solution}
8:    if MoveAcceptance(f, f') then
9:       S_best ← MaintainBestSolution(S'); {the new solution S' is maintained as the
         best solution S_best found so far}
10:      S ← S';
11:      f ← f';
12:   end if
13: until (termination criteria are satisfied);
14: return S_best;
```

by selection a random sequence of LLHs and apply them in order at each decision point [Cowling et al., 2000], and the *Random Descent (RD)* which randomly selects a LLH and keep applying it until it no further improves the solution [Cowling et al., 2001, Soubeiga, 2003].

Other hyper-heuristics apply more intelligent approaches which use some feedback from the search process when the algorithm is on the run. One of the best examples for these learning approaches is the "CF" [Burke et al., 2003, Gaw et al., 2004] which makes use of the information that the hyper-heuristics has access to, such as the CPU time used by each heuristic and the degree of improvement of the solution quality caused by each heuristic.

Reinforcement learning [Burke and Soubeiga, 2003, Fisher and Thompson, 1963, Nareyek, 2003] adaptively selects a heuristic from a set of pre-defined LLHs according to scores associated with each LLH. The scores increase and decrease as reward and punishment schemes, respectively. Through these reward and punishment schemes, the LLHs are rewarded by means of increasing their scores when they improve the current solution, and are punished by means of decreasing their scores when they do not improve the current solution. At each decision point, the heuristic with the maximum score is selected and applied on the current solution.

Two variations of probability-based heuristic selection method are employed in [García-Villoria et al., 2011]. In both variants, the probability of selecting a LLH is based on performance indicators. However, in the second variant of the probability-based heuristic selection method, tabu search method is used to exclude the relatively poor performing LLHs.

A selection method based on Markov chain model was developed by McClymont and Keedwell [2011b]. Other learning schemes include learning classifier system [Ross et al., 2002] and the Case based reasoning [Chakhlevitch and Cowling, 2008].

Other than these two approaches, researchers have used more sophisticated LLH selection approaches. For example, tabu search hyper-heuristic [Burke and Soubeiga, 2003, Kendall and Hussin, 2005] maintains a list to disable the selection of poorly performing LLHs. Other hyper-heuristics inspired by meta-heuristics are GAs [Cowling et al., 2002, Hart et al., 1998, 1999, Norenkov, 1994] and great deluge [Dueck, 1993]. Meta-heuristic based hyper-heuristics are shown to be very effective and often beat state-of-the-art problem-tailored methods [Chakhlevitch and Cowling, 2008].

More discussion on different selection strategies will be given in the following sections.

2.2.2.2 Move Acceptance Methods

The move acceptance is carried out using many different strategies as well. For example, in [Özcan et al., 2008], several different simple strategies are discussed. The (ALL) method accepts "All Moves" [Cowling et al., 2000], (OI) accepts "Only Improving" ones [Cowling et al., 2000], and (IEQ) accepts "Improving or Equal" moves [Bilgin et al., 2007].

SA [Burke et al., 2008] move acceptance method accepts improving moves by default. The non-improving moves are accepted according to a probabilistic formula, p, presented in Eq. (2.2).

$$p = e^{-\frac{\Delta f}{\Delta F(1-\frac{t}{T})}} \tag{2.2}$$

In this equation, T is the maximum number of steps, ΔF is an expected range for the maximum fitness change, and Δf is the fitness change at time or step t.

GDEL [Kendall and Mohamad, 2004] acceptance method attempts to escape from local optima by accepting moves that worsens the current solution within a range that is determined by a time-varying threshold, τ_t. This threshold starts with a certain value and decreases with time, as shown in Eq. (2.3).

$$\tau_t = f_0 + \Delta F \times (1 - \frac{t}{T}) \tag{2.3}$$

In this equation, f_0 is the final objective value. T and ΔF have the same definition as in Eq. (2.2) above.

A heuristic-selection hyper-heuristic can be created by combining any of the heuristic selection methods with any of the move acceptance methods discussed above; and the research question is hence to prove that a particular combination is superior to others. For example, The results of the study shown in [Cowling et al., 2000] indicate that the *choice function_all moves* combination is promising.

2.2.2.3 Selection Hyper-heuristic Frameworks

All of the hyper-heuristics discussed above treat all of the LLHs equally, and do not attempt to discriminate between them on whatever basis other than their performance. However, in [Özcan et al., 2008], the authors argue that it would be better to distinguish between the hill climbers and the mutational LLHs, and always enforce the application of a hill climber; in an attempt to mimic the mechanism of the Memetic Algorithms (MAs) [Goldberg, 1989, Holland, 1992].

FA is the traditional hyper-heuristic framework discussed earlier, where the level heuristic selected at a given decision point could be either mutational or a hill climber. FB, FC and FD are the proposed frameworks. The common idea between these three frameworks is that they all ensure that a hill climber is used at each decision point regardless to whether a mutational heuristic is used or not.

2.2.2.4 Hyper-heuristics Flexible Framework - HyFlex

In order to facilitate the development of hyper-heuristics, a software benchmark framework known as *HyFlex* [Burke et al., 2009a] has recently been developed to support the design of cross-domain heuristic search methods [Burke et al., 2011]. It provides a common interface to the algorithmic components for six selected hard combinatorial problems. Researchers only need to provide the high-level strategy that will manage and configure the provided algorithmic components. HyFlex has been used to support the cross-domain heuristic search challenge (www.asap.cs.nott.ac.uk/chesc2011). Twenty competitors have submitted their solvers, which thenceforth became a benchmarking tool for selection hyper-heuristics across different problem domains. The description of the competing algorithms and the results are provided in the competition website and briefly summarized in Table 2.2.

Table 2.2: The CHeSC 2011 competing approaches

Label	Score	Description [Reference]
AdapHH	181.00	Applied an adaptive heuristic selection combined with adaptive iteration limited list-based threshold move accepting method [Misir et al., 2013]
VNS-TW	134.00	Employed an approach based on variable neighborhood search which applies shaking (mutation and ruin&re-create) heuristics then hill-climber heuristics [Hsiao et al., 2012]
ML	131.50	Developed a self-adaptive meta-heuristic which uses multiple cooperating agents [Larose, 2011]
PHUNTER	93.25	Applied a method which aims to balance the diversification and intensification processes which inspired by the metaphor of pearl hunting [Chan et al., 2012]
EPH	89.75	Designed an evolutionary approach which co-evolves population of solutions as well as heuristic sequences [Meignan, 2011]
HAHA	75.75	Implemented a method switching between working on a single solution and a pool of solutions [Lehrbaum and Musliu, 2012]
NAHH	75.00	Developed a stochastic local search algorithm which consists of several schemata that have been tuned in an offline manner on four HyFlex problem domains [Mascia and Stützle, 2012]
ISEA	71.00	Employed an approach based on evolutionary iterative local search algorithms [Kubalík, 2012]
KSATS-HH	66.50	Implemented an approach based on simulated annealing, tabu search and reinforcement learning [Sim, 2011]
HAEA	53.50	Developed an approach based on evolutionary algorithms. The selection probability and the parameters of the heuristics are adapted [Gomez, 2004]
ACO-HH	39.00	Designed a solver based on the ant colony optimization method [Núñez and Ceballos, 2011]
GenHive	36.50	Employed a method inspired by the bee and genetic algorithms consisted in parallel search of the solution space using evolving sequences of heuristics [Cichowicz et al., 2012]
DynILS	27.00	Proposed an iterated local search variant, namely dynamic iterated local search which dynamically updates the selection probability of a heuristic [Johnston et al., 2011]
SA-ILS	24.25	Used a method based on simulated annealing
XCJ	22.50	Developed an approach based on classifier systems
AVEG-Nep	21.00	Implemented an unsupervised machine reinforcement learning. The learning is based on the 'trial and error' fashion to learn how to select the optimal action [Di Gaspero and Urli, 2012]
GISS	16.75	Developed a generic iterative simulated annealing search algorithm which makes use of a thermodynamical function in order to escape from the local optima [Acuña et al., 2011]
SelfSearch	7.00	Applied an online operator control algorithm namely 'self-search' at which the selection method is based on a several performance measures [Elomari, 2011]
MCHH-S	4.75	Developed a single objective variant of the online selective Markov chain hyper-heuristic [McClymont and Keedwell, 2011a]
Ant-Q	0.00	Used an agent based algorithm that combines ant colony based methods with the reinforcement learning technique for building sequences of LLHs [Khamassi, 2011]

2.2.3 Generation Hyper-heuristics

The work in [Burke et al., 2009c] in late 2000 used genetic programming for hyper-heuristics and led to the emergence of a new class of hyper-heuristics. A Genetic programming algorithm [Koza et al., 2003] is an evolutionary algorithm that takes two or more computer programs and applies ideas from the natural evolution theory such as inheritance (crossover), selection and variation (mutation) to produce new computer programs [Burke et al., 2009c].

Before this work, hyper-heuristics were generally defined as "heuristics to choose heuristics"; i.e. the hyper-heuristics framework was supplied with a set of pre-existing heuristics that are usually used to solve the target problem. In contrast, in the new class the idea is to generate new heuristics from the basic components of the set of the pre-existing heuristics supplied to the framework. More precisely, the pre-existing heuristics are first decomposed into their basic building blocks before been given to the framework [Burke et al., 2009b]. Examples of this approach can be found in [Bader-El-Den and Poli, 2007, Burke et al., 2006a, 2007a]. Also, researchers have made a distinction between heuristics that are evolved to be used for only one problem, which they call "*disposable*"; and those, which are evolved to be used on many - including unseen - problems of a certain class, which they call "*reusable*".

As a result of the emergence of this new class, the basic definition of hyper-heuristics has been modified to be: "*an automated methodology for selecting or generating heuristics to solve hard computational search problems*" [Burke et al., 2009b].

2.2.4 Framework-based Classification

A more recent classification divides hyper-heuristics into four categories based on the nature of their framework. The first one is (I) Hyper-heuristics based on the random choice of a LLH, from the set supplied to the framework, at each decision point. Approaches which fall in this class include the *Pure Random* hyper-heuristics which uniformly select LLHs and accept either all of them or the improving ones only. Examples are [Bai and Kendall, 2005, Burke et al., 2005, Cowling and Chakhlevitch, 2003, Cowling et al., 2000]. Also, this category include the *Random Descent* hyper-heuristics in which an improving LLH is applied repeatedly until it no further improves the solution such as in [Cowling et al., 2001, Soubeiga, 2003].

The second category is (II) Greedy and peckish hyper-heuristics which, at each decision point, select and apply the LLH which produces the largest improvement to the current objective value. An improved version of this approach attempts to prevent the

hyper-heuristic form getting stuck at local optima by allowing non improving LLHs to be applied. Clearly, hyper-heuristics of this category are quite slow since they perform initial evaluation of each LLH in the set in order to select the best one. Nevertheless, this is not the main disadvantage of hyper-heuristics of this category, but rather the fact that greedy hyper-heuristics do not fully explore the search space and they leave many promising regions unexplored [Chakhlevitch and Cowling, 2008]. An examples of a peckish hyper-heuristic is in [Cowling et al., 2000]. For the purpose of overcoming the problems associated with local optima, a group of "peckish" hyper-heuristics was proposed in [Cowling and Chakhlevitch, 2003] in which greedy and random mechanisms are combined for managing the choice of LLHs. Because of their generality and simplicity, greedy and peckish hyper-heuristics can be applied for different problems. However, they are unfavorable for problems where the time to construct solutions is a crucial factor due to their slow speed [Chakhlevitch and Cowling, 2008].

The third category is the (III) Meta-heuristic-based hyper-heuristics which comprises meta-heuristic approaches and their hybrids which have been used as high level heuristic selectors during the last few years. Most of the hyper-heuristics of this category are GA-based. GA-based hyper-heuristics give the solution indirectly, where each chromosome does not represent the solution itself, but rather the way a solution is constructed [Chakhlevitch and Cowling, 2008]. For example, in [Terashima-Marín et al., 1999], the chromosome represents a sequence of heuristics to be applied to the initial solution. Also in [Cowling et al., 2002], a GA-based hyper-heuristic approach is developed, called hyper-GA, in which an indirect GA operates at a higher level to evolve a sequence of LLHs from a given set, and then, the LLHs are applied in the order they appear in the sequence to find a good solution of the problem instance. Other examples are [Hart et al., 1998, 1999, Norenkov, 1994]. According to [Chakhlevitch and Cowling, 2008], indirect GAs are easy to implement and show robustness to problem changes.

The successful application of GAs in hyper-heuristic frameworks encouraged some researchers to use other types of meta-heuristics for the same purpose. The authors of [Kendall and Hussin, 2005] propose a simple tabu search based hyper-heuristic for solving examination timetabling problem. Each LLH becomes tabu, for a short and fixed duration, immediately after it is applied regardless to the quality of the solution it returns. Clearly, this simple straightforward approach is never able to beat the best known results for a range of benchmark timetabling problems. However, it found to produce good quality results provided a sufficient amount of CPU time is available. In [Kendall and Hussin, 2004], the same authors propose two improved versions of their tabu search based hyper-heuristic approach they suggested in [Kendall and Hussin, 2005]. The first version repeatedly applies the LLH which improves the previous best solution until it no longer improves it and becomes tabu. The second version borrows ideas from the

GDEL algorithm [Dueck, 1993] and accepts the best non-improving and non-tabu LLH only if it updates the solution within a certain boundary.

To sum up, meta-heuristic-based hyper-heuristics have been shown to be very effective, to the extent that they often beat state-of-the-art problem-tailored methods. The main problem with this approach is that different mate-heuristics require parameter fine-tuning; and moreover, parameters should naturally be altered to cope with different problems. This observation contrasts with the basic aim of hyper-heuristics; i.e. raising the level of generality. The ideal hyper-heuristic should be parameter-free in order to be applicable to new problems without significant amount of modifications and tuning [Chakhlevitch and Cowling, 2008].

The last category of this classification is (IV) Hyper-heuristics with learning. Hyper-heuristics of this group employ different techniques for learning from the historical performance of the LLHs. At each decision point, a LLH is chosen according to the information about its effectiveness accumulated in earlier stages of its run or in previous runs. Many hyper-heuristics such as Burke and Soubeiga [2003], Nareyek [2003], Soubeiga [2003] apply a popular learning mechanism known as the "Reinforcement Learning (RL)", the idea of which is to "reward" improving LLHs at each decision point and "punish" poorly performing ones by means of respectively increasing and decreasing their weights (scores) or probabilities of being selected [Chakhlevitch and Cowling, 2008].

Another famous learning mechanism used in many hyper-heuristics is a one that maintains an internal state and accumulates information about the recent performance of the LLHs. This information is accessed by a "CF" approach which decides on which LLH to be called next. In [Cowling et al., 2000], the Choice Function is defined as the "*key to capturing the nature of the region of the solution space currently under exploration and deciding which neighborhood (LLH) to call next, based on the historical performance of each neighbourhood*". For example, in [Cowling et al., 2000], the choice function is set to be the weighted sum of three components which reflect

1. Recent performance of each LLH,
2. Recent performance of pairs of heuristics, and
3. The amount of time since a given heuristic was last called, respectively.

The first two components aim to intensify the search while the third one is used to add some degree of diversification. The idea of the Choice Function is motivated by the observation that, the choice of a good LLH at any given time may be specified by the recent successful application of the heuristic or by the performance of this heuristic

in combination with another heuristic, or by the ability to redirect the search to other regions of the solution space to avoid a local optimum. The main disadvantage of this approach is that, for different problem domains, the weights of individual components in the Choice Function should be manually tuned to achieve the best results [Chakhlevitch and Cowling, 2008].

2.2.5 The Orthogonal Classification

The most recent classification of hyper-heuristics is proposed in [Burke et al., 2009b] as an attempt to provide a unified classification and definition that captures all the previous work in hyper-heuristics. This classification is based on

1. The nature of the heuristic search space, and
2. The source of feedback during learning.

According to the first dimension, there are two fundamental categories of hyper-heuristics, which are "heuristic selection" and "heuristic generation". According to the second dimension, there are two types of hyper-heuristics which are "online learning" and "offline learning" hyper-heuristics.

Selection hyper-heuristics based on constructive LLHs attempt to build a solution incrementally. Starting from scratch, they gradually build a complete solution by intelligently selecting and applying constructive heuristics. An example of online-selection-constructive based hyper-heuristics is [Burke et al., 2007c]. An example of offline-selection-constructive based hyper-heuristics is [Burke et al., 2006b]. On the other hand, selection hyper-heuristics based on perturbation LLHs attempt to iteratively improve an initially complete solution, which is found either randomly or by applying simple constructive heuristics. An example of hyper-heuristics of this category is [Cowling et al., 2000]. The application of a perturbation hyper-heuristic includes (I) selecting a LLH, and then (II) accepting or rejecting its output based on some strategy. The major difference between these two categories is that in the constructive based one there is a natural termination condition for the hyper-heuristic which is the point at which the solution is completely built. The perturbation based hyper-heuristics do not have such a natural termination criteria, and the process of selection new LLHs can be arbitrarily extended.

As discussed in section 2.2.3, most of the proposed generation hyper-heuristics use genetic programming [Burke et al., 2009c], some of which have evolved local search heuristics [Fukunaga, 2008], evolutionary algorithms [Oltean, 2005] and programs representing functions [Burke et al., 2007b]. Most of these hyper-heuristics are offline, and generate

reusable heuristics, since they are trained on some instances of the problem to generate heuristics, and then these heuristics are used on unseen instances of the same problem. There are also some examples of online generation hyper-heuristics which generate non-reusable or disposable heuristics since they are evolved for solving only one particular instance of the problem [Keller and Poli, 2008].

2.3 Summary

This chapter provided a brief literature survey of heuristic search methods, covering the well-known meta-heuristics as well as hyper-heuristics, and highlighting the recent developments in selection hyper-heuristics and cross domain search methodologies. The following chapter provides a literature survey of grouping problems, with a critical discussion about the main grouping solution representations and associated search operators used in the scientific literature, along with a survey of selected grouping problem domains.

Chapter 3

Grouping Problem

In this chapter, a survey of grouping problems is provided, focusing on the main solution representation schemes and associated operators proposed in the scientific literature. Firstly, a formal description of the grouping problem is presented. Thereafter, five well-known solution representations for grouping problems, each associated with relevant search operators/heuristics are discussed. These solution representations include Numeric Encoding (NE), Object Permutations (also known as the Group Encoding (GE)), Locus-based Adjacency (LBA), Linear Linkage Encoding (LLE) and Grouping Genetic Algorithm Encoding (GGAE). Moreover, a literature review of three grouping problem domains, namely Graph Coloring, Exam Timetabling and Data Clustering which are used as selected case studies to illustrate the performance of a range of selection hyper-heuristics, implemented based on the proposed framework.

3.1 Problem Definition

As suggested by its name, the aim in grouping problems is to partition the objects of a set U into a minimal collection of mutually disjoint subsets u_i of U, such that each object is in exactly one subset. An essential fact about this definition is that it is based on the composition of the groups; i.e. single objects have almost no meaning when taken in isolation. Hence, the maximum number of groups a solution to a grouping problem can have must always be less than the number of items in U. Additionally, different grouping problems have different problem specific constraints, and introduce different objective (cost/penalty/fitness) functions. In graph coloring [Hertz and Werra, 1987] for example, connected nodes are not allowed to be in one group. Also, bin packing [Coffman et al., 1997], the sum of the sizes of the objects in a given group must not exceed a given upper limit, c. Consequently, not all groupings are allowed for all

problems, since a solution must satisfy the problem specific constraints. These problem specific constraints/objectives forbid the 'trivial' solutions which consist of placing all the objects into one group.

Based on the above discussion, grouping problems can be formally defined as a class of combinatorial optimization problems in which a large group U of n items, $U = \{x_1, x_2, x_3, ..., x_n\}$, is to be divided into a collection of k ($2 \leq k \leq n-1$) subgroups, u_i ($1 \leq i \leq k$); such that each item $x \in U$ belongs to exactly one subgroup; while minimizing a given objective (cost/penalty/fitness) as well as the number of groups k. In the formulation given below, a cost function is denoted as a decomposable function, $f()$. For a subgroup u_i, the partial cost is denoted as $f(u_i)$, and for a complete solution $U_g = \{u_1, ..., u_k\}$, $f(U_g)$ is the total cost.

$$\textit{minimise} \qquad Z = (k,\ f(U_g)) \tag{3.1}$$

$$\textit{where} \qquad f(U_g) = \sum_{i=1}^{k} f(u_i) \tag{3.2}$$

$$\textit{subject to} \qquad \bigcup_{i=1}^{k} u_i = U \tag{3.3}$$

$$u_i \bigcap u_j = \emptyset \qquad \forall_{i,j} \text{ where } i \neq j \tag{3.4}$$

$$u_i \neq \emptyset \qquad \forall_i \tag{3.5}$$

Although grouping problems may seem to be easy to solve, most, if not all, of them are found to be NP-hard. A problem is NP-hard when every problem in NP can be reduced in polynomial time to the NP-hard problem [Falkenauer, 1998]. Informally, this implies that the time to run an exact algorithm on a grouping problem would be exponential in the size of the problem instance.

In this study, the grouping problems are formulated as bi-objective discrete multi-criteria problems in which the goal is to optimize the two conflicting objectives in Eq. (3.1) above, namely the number of groups which can only take discrete values; and the cost function which can take discrete or continuous values depending on the nature of the problem being solved. Ideally, those two objectives should be simultaneously optimized, although, in this case, they are conflicting; i.e. optimizing one of them will lead to the deterioration of the other. A decrease in the number of groups k means that more items are grouped together in fewer groups, and hence, leads to an increase in the cost. In some cases, there might not be a single optimal solution. Instead, there could be multiple solutions with a trade-off from which a decision maker can choose. Those solutions are

identified using the concept of *dominance* [Zitzler and Thiele, 1998] as illustrated in Figure 3.1.

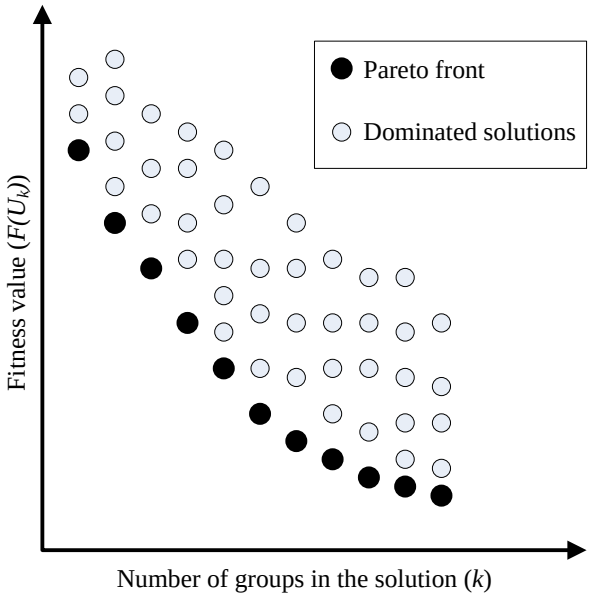

FIGURE 3.1: The dominance concept in multi-objective optimization.

Generally, a solution s_i is considered to dominate another solution s_j, ($s_i \succ s_j$) if, and only if, s_i is better than s_j in at least one objective, and s_i is not worse than s_j in any of the other objectives. The set of the non-dominated solutions is known as the *Pareto optimal set*, and its image in the objectives domain is known as the *Pareto optimal front* [Zitzler and Thiele, 1998]. This problem is different from a generic multi-objective problem where mostly, there is a region where the Pareto front is driven automatically via a multi-objective algorithm. However, in grouping problems the range of groups is fixed, hence the search methodology can focus on a single objective without ignoring the second one. We use some basic ideas from multi-objective optimization, but the proposed approach is not a generic multi-objective algorithm, as described in Chapter 4.

3.2 Representations

Many previous approaches to grouping problems are population based approaches [Falkenauer, 1998]. Hence, a variety of encoding schemes and genetic operators has been proposed and applied to various grouping problems since the emergence of the Genetic Algorithms Holland [1992]. However, most of these approaches were confronted by two main challenges that have to be carefully addressed when designing a solver for grouping problems. Namely, (i) the choice of the genetic representation (encoding) scheme

of the grouping solutions, and (ii) the design of the associated genetic/search operators or (heuristics) that are used to sample the search space starting from the current solution(s). According to Falkenauer [1992], grouping representations and associated operators should be designed such that they are *aware* of the nature of the grouping problem being solved, as well as of the ultimate objective of the grouping problems, which is to construct groups that maximize the fitness values of the solutions. Inappropriate solution representations and/or poorly designed operators could significantly damage the good solutions while they are being developed during the search, and hence, impair the grouping algorithm considerably.

Various research efforts have addressed these challenges by suggesting different encoding schemes and operators that vary in their qualities and abilities when used with grouping problems. Most of these representations suffer from degeneracy, which happens when a genetic representation allows multiple chromosomes to represent the same solution [Radcliffe and Surry, 1994]. Falkenauer [1998] uses the term "redundancy" to describe this problem, but Tucker et al. [2005] point out that degeneracy and redundancy are not the same thing, and suggest that redundancy refers to the "amount of excess information in the chromosome". In this thesis, Falkenauer's term is adopted since there is no fear of confusion, and hence, the term redundancy is used to describe the problem of allowing multiple chromosomes to represent the same solution. A grouping representation with redundancy can lead to the creation of a huge search space that will take a long period of time to be explored, because the the same configuration of groups are repeatedly visited.

3.2.1 Representing the Objects Membership

The first grouping representation that was explored in the scientific literature is the objects membership representation, which is also known as the numeric encoding "NE" representation [Ding et al., 1994, Talbi and Bessiere, 1991, Van Driessche and Piessens, 1992]. This straightforward representation scheme uses constant-length chromosomes in which each gene corresponds to one object, such that the value of that gene determines the group to which the corresponding object belongs. Consequently, the length of the chromosome is equal the number of the objects to be grouped.

1	2	3	4	5
C	D	F	F	D

(3.6)

To illustrate, the chromosome shown in Figure 3.6 represents a solution in which the first object is in group C, the second and fifth objects are in group D, and the third

and the fourth objects are in group F. Clearly, one of the main advantages of this representation is that it is very simple and straight forward. Also, the fact that the length of the chromosomes is constant facilitates the application of most of the standard genetic crossover and mutation operators.

The Redundancy in the Numeric Encoding

First of all, it is clear that the NE representation is highly redundant, since any solution can be represented by many different points in the search space. For instance, if we consider the chromosome shown in Figure 3.6 to be a solution for a bin packing problem of five objects, then the same solution could be represented by a completely different chromosome such as the one given in Figure 3.7:

1	2	3	4	5
A	B	C	C	B

(3.7)

The reason for this is that both of these two chromosomes actually indicate that the first object is in a group, the second and the fifth objects are in another group and the third and the fourth objects are in a third group. Obviously, changing the names of the groups does not make any difference; what really matters is the actual grouping of the objects. This problem violates the "minimal redundancy" principle, which is one of the six design principles for constructing useful representations stated by Radcliffe [1991]. This principle states that in order to decrease the size of the search space that the algorithm has to explore, each solution should be represented by as few distinct points in that search space as possible; ideally one point only. However, as has been shown above, the NE representation causes the search space to grow exponentially with the number of possible groups; that is, indirectly with the number of objects.

Standard Crossover: Context-insensitivity and Schema Disruption

Another problem with this representation is the problem known as the "context insensitivity" [Falkenauer, 1992]. Considering both of the two chromosomes above, it can clearly be seen that each one of the letters that are used to identify the groups has a "meaning" only within the context of the particular chromosome in which it is used. To explain this further, the letter C in the first chromosome shown in Figure 3.6 encodes something that is completely different from what the same letter encodes in the second chromosome given in Figure 3.7. The C in the first chromosome merely means that the first object is not grouped with any other object, while in the second chromosome,

the same letter means that the second and the third objects are grouped together in a distinct group. However, this fact is completely ignored by the NE representation; and the associated operators "know" nothing about it and, as a result, are insensitive to the contextual meanings of the letters. The effect of this insensitivity can be extremely destructive. For instance, when the genes are exchanged between two chromosomes during the crossover, the letters are taken out of context and are put together in the recombination process, which produces a highly "distorted" and meaningless offspring.

This last point can be easily manifested by the following simple example. It is well known that when applying the crossover operator to any two identical parents, the resulting offspring should be identical to each one of the parents. However, if we take the two chromosomes above again, which were shown to represent the same solution, and apply a standard single point crossover as shown in below:

1	2	3	4	5
C	D	F	F	D

(3.8)

1	2	3	4	5
A	B	C	C	B

(3.9)

One of the resulting offspring would be

1	2	3	4	5
C	D	C	C	B

(3.10)

The chromosome shown in Figure 3.10 represents a solution which is completely different from the one that is represented by either one of the parents shown in Figure 3.8 and Figure 3.9. This example proves that, the context sensitivity problem seriously impairs the operators and prevents them from achieving good performances and improving the current solutions during the search.

Along the same lines, applying the standard crossover on the numeric encoding can lead to a problem known as the "schema disruption" [Falkenauer, 1992]. In problems of big sizes, the corresponding NE chromosomes are too long since their length is equal to the number of objects given in the problem. The standard crossover might begin the search by converging to good solutions in which more objects are grouped together in a fewer number of groups. However, as the search progresses, the crossover is very likely to be very destructive and destroys the good solutions instead of improving them.

Standard Mutation in the Numeric Encoding

Finally, applying the standard mutation on the numeric encoding has its disadvantages too. As discussed in section 3.2.1, the cost function of the grouping problems is based on minimizing the number of the groups. If, for example, a valid solution distributes the objects into three groups, then another solution that distributes the same objects into four groups would simply be considered a bad solution. However, the standard mutation is not aware of this fact and might easily damage the good solutions when applied. For instance, the chromosome

1	2	3	4	5
C	D	F	F	D

(3.11)

distributes the objects into three groups. Applying the standard mutation could simply result in introducing a new group

1	2	3	4	5
C	D	F	G	D

(3.12)

Which might be argued to be beneficial since a new allele (group) emerges in the population. However, this solution is highly likely to be eliminated from the population in the very next step of the algorithm, gaining nothing but a serious delay in the search.

3.2.2 Modified Operators for the Objects' Membership Representation

Based on the above discussion about the issues associated with the objects membership representation for grouping problems, an attempt to capture and reflect the structure of the grouping problem in the design of the operators in a better way has been carried out by von Laszewski [1991]. Laszewski used the same standard object membership representation explained in section 3.2.1, but new "intelligent operators" were designed to work with that representation.

The main observation behind their work is that in order for the crossover to make sense in the context of grouping problems, it must transmit "whole groups" rather than separated objects. In other words, the genetic operators should be "group oriented" rather than "object oriented". Hence, von Laszewski [1991] designed a special crossover operator that works as explained below.

1. Select one whole group from the first parent to be copied into the second parent.
2. In the receiving parent, select the group that is most similar to the group that has been copied in the previous step. The similarity between two groups is defined in terms of the number of object they have in common.
3. Since the exchanged parts should be of the same length, modify the groups in the receiving parent so that the most similar group identified in the previous step has the same number of objects as the group that was copied in the first step.
4. Remove the most similar group from the receiving parent and copy it to the first parent.

Comments on the Modified Numeric Encoding Operators

The main breakthrough in this modified approach is the highlighting of the fact that special operators that are aware of the structure of the grouping problems should be designed to handle them. However, this on its own is not enough to completely capture the problem structure. Clearly, it should be supported by a problem-aware encoding as well. Another issue with this approach is that it limits the crossover to transmit only one group at a time, and it does not clearly specify how to select the groups to be transmitted nor how the groups in the receiving parent should be modified as mentioned in step 3 [Falkenauer, 1998].

3.2.3 Representing Objects Permutation

This representation scheme, which also known as the Group Encoding representation "GE", is an alternative representation scheme that was adopted by many researchers in the scientific literature [Reeves, 1993, Smith, 1985]. Similar to the NE, this scheme also uses a 'one gene per object' approach, but the chromosomes are interpreted differently. The idea in this approach is to represent the objects permutations rather than their memberships. The GE representation is concerned with the *positions* of the genes, or more precisely, their positions relative to each other; and the actual values of the alleles no longer matters. The chromosomes are viewed as encoded solutions since they only represent objects permutations. A decoding heuristic is needed in order to obtain the actual solutions. In other words, the actual distribution of the objects into groups can not be seen directly from the GE chromosome. For example, if we consider a bin packing problem with 5 objects, the following chromosome

Objects permutation:	5	3	1	2	4
Actual groups:	?	?	?	?	?

(3.13)

would represent a valid permutation of the five objects given. However, in order to obtain the actual grouping of these objects, a decoding mechanism should be applied. An example of such a decoding mechanism could be the following: "starting from left to right, put each object into the last open bin if that object can be accommodated in that bin; otherwise, request a new bin". Applying this decoding heuristic to the above chromosome could result in the following actual grouping of the objects:

Objects permutation:	5	3	1	2	4
Actual groups:	A		B		C

(3.14)

which can now be interpreted as that items 5 and 3 are in one group, items 1 and 2 are in another group and item 4 is in a third group on its own.

Again, it is obvious that this representation is highly redundant for similar reasons as those discussed in section 3.2.1. Many chromosomes, or objects permutations that are completely different from the one shown in Figure 3.13, could yield the same solution or actual grouping as the one shown in Figure 3.14. For instance, any permutation of any two of the groups such as in the chromosome shown in Figure 3.15, will still give the same solution as before. Consequently, similar to the NE, using this representation scheme would allow the number of distinct chromosomes encoding the same solution of the original problem to grow exponentially with the number of groups. This will seriously impair the search algorithm, since the size of the resulting space that the algorithm has to explore becomes much larger than the original space of solutions.

Objects permutation:	2	1	5	3	4
Actual groups:	A		B		C

(3.15)

Ordering Crossover: Context-insensitivity and Schema Disruption

Moreover, the objects permutation representation also suffers from the context insensitivity problem when used with any standard genetic operator, including the standard ordering crossover that can be successfully used to transmit the absolute positions of the genes. As discussed by Falkenauer [1998], using the standard ordering crossover with this representation could have a disruptive effect on the good solutions while they are being developed during the search. This could be clearly explained by highlighting two

facts. Firstly, taking the same decoding algorithm described above, the position of a gene only makes sense when interpreted in relation to the other genes that precede it in the chromosome, and more precisely in relation to the members of other preceding groups. Manipulating the positions of the genes in these preceding groups would result in taking the value of that gene out of context. To illustrate, in the following chromosome

Objects permutation:	5	3	1	2	4
Actual groups:	A		B		C

(3.16)

objects 1 and 2 are in the same group. This fact depends on the objects that precede them in the chromosome, i.e. objects 5 and 3. Manipulating any of these objects such as in the following two chromosomes

Objects permutation:	4	3	1	2	5
Actual groups:	?	?	?	?	?

(3.17)

or

Objects permutation:	4	5	1	2	3
Actual groups:	?	?	?	?	?

(3.18)

would yield two completely different solutions than the first one; and no guarantee that the two items 1 and 2 will any longer be in the same group.

The second fact is that, as much as the ordering crossover used with this representation could be very successful in transmitting the absolute positions of the genes, it will be absolutely insensitive to their relations to each other. This could be very destructive, since the relative meaning of the transmitted information need to be considered. As a result, these two facts put a big question mark on the validity of the whole approach. Similarly, the standard ordering mutation operator could have destructive effects when used with the permutation representation, because it modifies the order of the genes within the chromosome. Clearly, this modification affects all of the genes that follow the modified gene, and consequently, all the values of these genes will be taken out of context [Falkenauer, 1998].

3.2.4 The Linear Linkage Encoding "LLE"

In [Park and Song, 1998], a linkage-based representation has been proposed, known as the Locus-Based Adjacency (LBA) representation, which is a fixed-length representation. In the LBA, each location represents one object and also stores an integer value that represents a link to *another* object in the same group. Starting from any object in a given group, the links are used to reach to all of the other objects within that group. This representation has been successfully applied in many grouping and data clustering problems. For instance, Handl and Knowles [2007] designed an evolutionary algorithm for multi-objective data clustering problems using the locus-based adjacency representation.

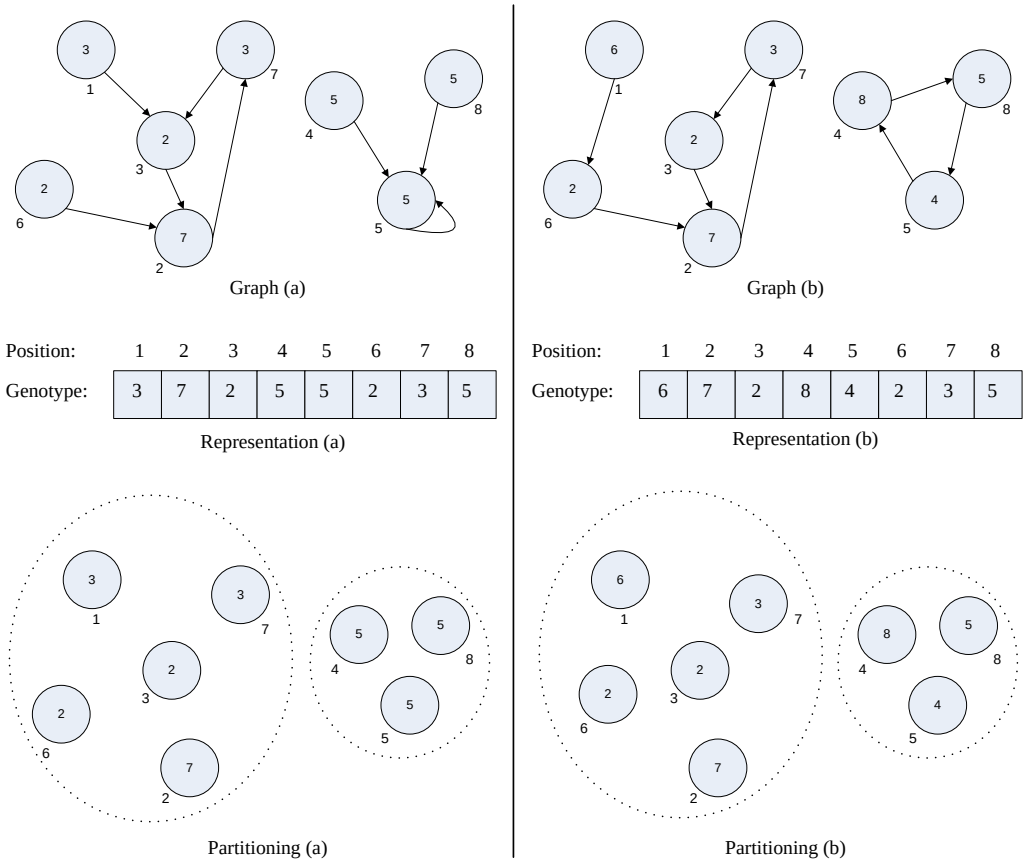

FIGURE 3.2: Two grouping solutions (a) and (b) encoded using the locus-based adjacency (LBA) representation. Although the graphs that represent the two solutions are different, the groups in the two solutions are exactly the same: 1, 2, 3, 6, 7 and 4, 5, 8

The LBA representation is flexible and allows for a direct application of most of the standard genetic operators, but it still suffers the same issues of redundancy and lack of appropriate grouping-oriented operators as the former representations discussed above, since any gene in a given cluster can point to any other gene in the same cluster. In the example given in Figure 3.2, This clearly means that, any given group can be represented by multiple chromosomes that can be produced by any permutation of the links.

The LLE is a recently proposed encoding scheme which was firstly used for data clustering [Du et al., 2004] and then applied for graph coloring and timetabling as well as other grouping problems [Ülker et al., 2008, Ülker et al., 2006]. LLE is a restricted form of the more general Linkage encoding (LBA) scheme. According to Du et al. [2004], an LBA chromosome is considered to be a valid LLE chromosome if it satisfies the following two conditions:

1. The integer value stored in each gene is greater than or equal to its index but less than or equal to the number of the objects.
2. Other than the index of an ending node, no two genes can have the same value.

These two conditions imply that, backward links are not allowed in the LLE representation. Also, each node must have only one node pointing to it, apart from an ending node of a group which contains its own index and points to itself. A grouping solution encoded using the LLE representation is shown in Figure 3.3. In this solution, there are nodes that contain their own index, which are nodes 4, 7 and 8. This indicates that this solution contains 3 groups, namely 1, 2, 5, 7, 3, 6, 8 and 4. Unlike the LBA encoding, the chromosome shown in Figure 3.3 is the only valid LLE chromosome that can represent this solution.

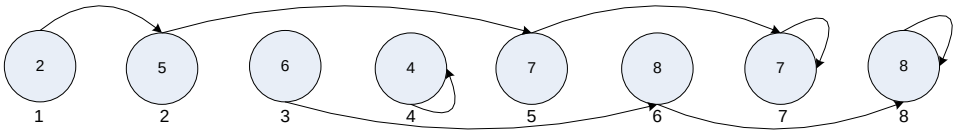

FIGURE 3.3: A grouping solution encoded using the linear linkage encoding (LLE) representation.

Using the restrictions mentioned above, the LLE representation successfully eliminates the redundancy problem that debilitates former representations. The proposed LLE encoding has the advantage of making a one-to-one mapping between the LLE chromosomes and the actual grouping solutions. The LLE encoding identifies the membership of a given object using either the starting or the ending node of the cluster. Practically, finding either of these nodes takes only linear time.

LLE Crossover

LLE crossover operators should be designed in such a way that maximizes the advantage of the non-redundant search space of the LLE representation. Basically, many of the well known crossovers could achieve this goal. For instance, in the original LLE proposal

[Du et al., 2004], a simple one point crossover was adapted. However, in some cases, problem specific information might be taken into account in order to make the search non-redundant in addition to the representation. The work shown in Ülker et al. [2006] proves that the simple one point crossover performs poorly when applied on the graph coloring problem due to the danger of introducing new links that were not available in both parents. Instead, the authors use three different types of crossover that have been tailored to suite the problem at hand.

LLE Mutation

Many simple mutation operators can be designed for the LLE representation. Since the genes of an LLE chromosome are linearly linked to each other, changing the value of one gene could not only result in modifying the group of one item, but actually changes the groups of all of the items that precedes that gene. Moreover, it could also result in no changes at all if the new gene value points to the same group again. The main guideline when designing an LLE mutation operator is to make sure that the new out link points to a different group, not only a different node. The design may address other issues as well, such as the fact that changing the value of one gene re-assigns many objects instead of only one as in the case of other representation schemes.

There are several studies that use LLE for solving grouping problems. Korkmaz [2006] used LLE to solve two-level clustering problem with a genetic algorithm and used large data sets. Ülker et al. [2006] worked on two grouping problem domains, Graph Coloring and Timetabling. Özcan et al. [2009] worked on Examination Timetabling problem with a newly proposed acceptance criteria Late Acceptance (LACC) [Burke and Bykov, 2008]. Ülker et al. [2008] used LLE for solving bin packing problems with grouping genetic algorithm. The main criticism to the LLE representation is that the computational time required to maintain the LLE definition during the search process is very high, particularly while using some genetic operators [Yılmaz and Korkmaz, 2010]. Two supplementary versions of LLE referred to as Linear Linkage Encoding With Ending Node Links (LLE-e) and Linear Linkage Encoding With Backward Links (LLE-b) are used in [Yılmaz and Korkmaz, 2010] in combination with the standard LLE representation. When the cost of applying a particular operator on an LLE chromosome is high, the operator is applied on one of the supplementary versions and the results are reflected back on the LLE chromosome.

3.2.5 The Grouping Genetic Algorithm Encoding "GGAE"

The GGAE is a representation used as part of a genetic algorithm that is heavily modified to suit the structure of the grouping problems. This algorithm was invented by Falkenauer [1992] and has been successfully applied in many real world problems since then, such as data clustering [Agustın-Blas et al., 2012] and machine-part cell formation [Brown and Sumichrast, 2003].

The main observation on which this algorithm was built is that all of the previous representations adopted by the researchers were not able to accurately capture the structure of the grouping problems. This is mainly due to the fact that, these representations were all object-oriented rather than group-oriented. In other words, the previous representations do not directly represent the essence of the cost of function of the grouping problems which is based on the construction of groups rather than the single objects. The GGAE attempts to address the issues discussed in sections 3.2.1 and 3.2.3 by introducing a new representation that is mainly tailored for the grouping problems, as well as groups-aware operators.

GGAE Representation

The chromosome in the GGAE consists of two parts. The first part is similar to the standard object membership encoding described in section 3.2.1. However, in the GGAE, this part is used only to identify which objects are actually in which groups. No operators are applied to this part of the encoding at all.

The second part is the groups part that encodes the groups on a "one gene per group" basis. In the illustrations below, the group part is written after the standard numeric encoding part, and the two are separated by a colon. For example, a solution that puts the first three items in one group and the last two items in a different group is represented as shown in Eq. (3.19) below.

1	2	3	4	5	:	G1	G2
D	D	D	C	C	:	D	C

(3.19)

In the GGAE, it is the groups part that is manipulated by the search operators. Consequently, applying the genetic operators on a GGAE chromosome might lead to the introduction of one or more new groups or the disappearance of one or more others. Clearly, this approach implies that the length of the GGAE chromosomes is not fixed and can not be known in advance, since the number of the groups varies from one solution to the other. Accordingly, the operators will be working on varying length chromosomes.

This representation, in its current form, is also redundant as argued by Tucker et al. [2005]. It can easily be seen that the solution shown in Eq. (3.19) could be represented by multiple other chromosomes such as the two shown in Eq. (3.20), where the group identification letters are swapped, and 3.21, where completely different groups identification letters are used. Clearly, any permutation of any two group identification letters would result in a chromosome that represents the exact same solution.

1	2	3	4	5	:	G1	G2
C	C	C	D	D	:	C	D

(3.20)

1	2	3	4	5	:	G1	G2
B	B	B	A	A	:	B	A

(3.21)

A slightly modified version of the GE representation that is structured to overcome the redundancy issue is commonly used. Two well-known restrictions are introduced to the standard Falkenauer's GGAE representation in order to eliminate this redundancy:

- In any chromosome with k groups, groups must be enumerated using values 1 through k (or k letters starting from A), only.
- Groups containing items with lower indices are enumerated first; i.e. the first item must always be placed in group 1 (or A). The next item that belongs to a different group must be place in group 2 (or B), and so on. This implies that the last item must always be placed in the k^{th} group.

Applying these two restrictions, the chromosomes shown in Eq. (3.19), Eq. (3.20) and Eq. (3.21) can only be represented as shown in Eq. (3.22) below.

1	2	3	4	5	:	G1	G2
A	A	A	B	B	:	A	B

(3.22)

GGAE Crossover

As well as introducing a new encoding method, the operators that work with the GGAE are also designed such that they are aware of the nature of the problem being solved. The GGAE crossover differs from the standard crossovers not only in the fact that it manipulates varying length chromosomes, but also by being aware of the nature of the problem, not just act as a cut-and-concatenate operator. The GGAE crossover that was suggested by Falkenauer [1998] adapts some groups that are present in one of the

parents and copies them into the other parent. The details of the working mechanism of this crossover vary from one grouping problem to another, but it should always perform certain steps. For example, considering the following groups parts from two parents:

:	G1	G2	G3	G4	G5	G6	G7
Groups Part of Parent1:	A	B	C	D	E	F	G
Groups Part of Parent2:	a	b	c	d	e		

(3.23)

The common steps of the GGAE crossover are as follows:

1. Within the groups part of the each one of the two given parents, select two random crossing sites.

:	G1	G2	G3	G4	G5	G6	G7
Groups Part of Parent1:	A	B |	C	D	E |	F	G
Groups Part of Parent2:	a |	b	c	d |	e		

(3.24)

2. Inject the groups between the two crossing sites of the second parent into the first crossing site of the first parent.

:	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Groups Part of Child1:	A	B	b	c	d	C	D	E	F	G

(3.25)

3. In the receiving parent, some objects are now members of more than one group. Get rid of the groups that contain any of the copied objects, leaving the groups that were copied in the previous step intact.

:	G1	G2	G3	G4	G5	G6	G7
Groups Part of Child1:	A	B	b	c	d	E	F

(3.26)

"assuming that groups C, D and G contain objects that are also available in the copied groups".

4. Now, eliminating the groups that include common items with the injected groups will also lead to the elimination of some other items that were not in the injected groups. These items have to, somehow, be "reinserted" into the groups that are still available in the receiving parent. It is this reinsertion step that is highly dependent on the particular grouping problem being solved. In fact, the choice of the reinsertion heuristic in this step greatly affects the performance of the genetic algorithm, as discussed in the following section.

5. The second child is generated by reversing the transmitter and the receiver parents and repeating the same steps as above.

The Impact of the Reinsertion Heuristic on the Performance of GGAE Crossover

The fourth step of the GGAE crossover operator uses reinsertion heuristics that are dependent on the particular grouping problem being solved. It has been argued that the performance of the GGAE crossover largely depends on the choice of the reinsertion heuristics. Brown and Sumichrast [2003] compared the performance of a crossover that uses a naive reinsertion heuristic to a crossover that uses an intelligent one. They applied both operators to some problem instances from the Machine-Part Cell Formation "MPCF" problem. The naive reinsertion method places the unassigned objects into groups in a random fashion. On the other hand, the intelligent approach makes use of some problem specific information to place the unassigned objects into the most suitable groups; i.e. to place each unassigned component into the group with the most machines it needs, and each unassigned machine into the group with most components that require it. They evaluated the results and found that the performance of the genetic algorithm could be improved by up to 40% when incorporating problem-specific knowledge into the reinsertion heuristic.

GGAE Mutation

Again, the standard mutation operators discussed earlier are too destructive due to the object-oriented representations they were applied to. A good grouping mutation operator should work with the groups rather than the objects themselves.

As is the case with the GGAE crossover, the exact details of the GGAE mutation operator depend on the particular problem being solved. However, according to Falkenauer [1998], three general alternatives can be stated: create a new group, eliminate an existing group or shuffle a small number of objects between groups.

3.3 Grouping Problem Domains

3.3.1 Graph Coloring Problem

Graph coloring is an important combinatorial optimization problem in which the task is to assign colors or labels to certain elements of a graph subject to certain constraints.

The research presented in this thesis focuses on a well-known variant of graph coloring known as the vertex coloring problem, in which, a minimal number of colors must be used such that connected nodes are of different colors. There are other well-known variants of graph coloring such as the edge coloring problem in which edges that are incident to the same node must be of different colors. The minimum number of colors that can be used to color a given graph G is referred to as the *chromatic number* of G, denoted as $\chi(G)$. Graph coloring has been applied to a wide variety of problems in many different fields such as data mining, image capturing and segmentation, physical layout segmentation, radio frequency assignment and GSM mobile phone networks, to name a few [Hertz and Werra, 1987, Jensen and Toft, 1994, Leighton, 1979].

Mathematically, given a an undirected graph $G = (V, E)$ with a set of vertices $V = \{x_1, x_2, x_3, ..., x_n\}$, and a set of edges E, where $e_{x_p,x_q} \in \{0, 1\}$ represents whether there is an edge between two given vertices (1) ($x_p, x_q \in V$) or not (0), graph coloring problem (GCP) requires coloring of vertices using a given number of colors such that none of the adjacent vertices (connected with an edge) are in the same color, hence *conflict free*. Determining the chromatic number of a given graph G, and finding out whether G is k-colorable (whether k colors are sufficient to create a conflict free graph) are well-known variants of the GCP which are NP-hard and NP-complete, respectively [Saha et al., 2013].

In this study, a minimum coloring variant of GCP is formulated as a multi-criteria grouping problem. Assuming that the number of colors is denoted as k, where $2 \leq k \leq n-1$, and v_i represents a subgroup of V in which the vertices are assigned to the same color i, $(1 \leq i \leq k)$, the definition of the grouping problem defined in section 3.1 follows, taking $colours \Leftrightarrow groups$, $V \Leftrightarrow U$, and $v_i \Leftrightarrow u_i$. The cost can be measured using Eq. (3.27).

$$f(u_i) = f(v_i) = \sum_{p,q} e_{x_p,x_q} \quad , \forall x_p, x_q \in v_i \;\; and \; e_{x_p,x_q} \in \{0, 1\}, where \; p < q \tag{3.27}$$

Graph coloring is one of the extensively studied areas of research, yet many studies have been emerging. Many techniques have been proposed for solving many variants of GCP. Exact approaches tend to fail particularly while solving *large* instances, hence many researchers have been working on heuristic approaches. For example, recursive largest fit is a well-known greedy heuristic introduced by Leighton [1979]. Hertz and Werra [1987] presented the first tabu search implementation which outperforms another local search method, simulated annealing, on random dense graph instances. Davis and Mitchell [1991] proposed a coding as an ordering of vertices which could be used in a genetic algorithm. Johnson et al. [1991] presented three simulated annealing implementations

based on three neighboring approaches. Galinier and Hao [1999] have shown that hybridization of GAs with local search methods are more promising than GAs on their own. In such hybridization, local search operators are used as intensification methods to explore promising areas of the search space that have found by the GA operators. Avanthay et al. [2003] proposed a variable neighborhood search algorithm for graph coloring problem.

Studies presented in [Kirovski and Potkonjak, 1998, Korkmaz, 2010, Ülker et al., 2008, Ülker et al., 2006] apply generic GAs using some of the genetic representations discussed in section 3.2 to solve graph coloring problem. Ülker et al. [2006] proposed special crossover operators for graph coloring, namely Lowest Index Max Crossover (LIMX), Greedy Partition Crossover Lowest Index (GPX-LI) and Greedy Partition Crossover Cardinality Based (GPX-CB). Yılmaz and Korkmaz [2010] proposed two modified versions of the LLE representation which are the LLE-e and the LLE-b, and both of them are tested using genetic operators.

3.3.2 An Examination Timetabling Problem

Graph coloring underpins a variety of real-world problems. Examination timetabling is one of those problems which requires allocation of periods/time-slots and other available resources to a given set of examinations taken by a number of students subject to a set of certain constraints. This problem can be formulated as a graph coloring problem, considering that the examinations are the nodes of a graph while the edges are formed using the the examinations that will be taken by each student, since those examinations taken by each student cannot be allocated the same time-slot (color). Johnson and Trick [1996] showed that the exam timetabling problem can be reduced to graph coloring problem if the task of minimizing the number of exam periods and removing the clashes are considered.

There are many variants of examination timetabling problems. Again, many different heuristic approaches have been the focus of previous studies due to inherent difficulty in solving timetabling problems, ranging from ordering of graph coloring heuristics [Carter and Laporte, 1996] to evolutionary approaches [Burke and Newall, 1998, Paquete et al.]. The same formulation of an examination timetabling problem in Carter et al. [1996] is used in this thesis. The relevant instances are identified as *Toronto a* in Qu et al. [2009]. Caramia et al. [2001] proposed a family of local search-based timetabling algorithms that apply an optimization step after each exam allocation attempting to minimize the number of time slots (groups) and the overall penalty simultaneously. Merlot et al. [2002] presented a hybrid algorithm in which a hill climbing phase and a simulated annealing

phase are used to improve an initial solution that was developed using a constraint programming phase. Ülker et al. [2006] formulated the problem as a grouping problem and used the LLE encoding presented in section 3.2.4 to test a variety of evolutionary approaches on this problem. More on examination timetabling can be found in Carter et al. [1996] and Qu et al. [2009].

3.3.3 Data Clustering Problem

Data clustering problem requires partitioning a given set of data items or property vectors into a minimal number of disjoint clusters/groups, such that the items in each group are *close* to each other with respect to a given similarity measure, and are *distant* from the items in the other groups with respect to the same measure. Data clustering plays an important role in many disciplines where there is a need to learn the inherent grouping structure of data such as data mining, pattern recognition, machine learning and bioinformatics [Agustın-Blas et al., 2012, Jain et al., 1999, Mitra and Banka, 2006, Park and Song, 1998]. Mathematically, a data clustering problem can formulated as follows. Given a set X of n vectors in a given feature space S, $X = \{x_1, x_2, x_3, ..., x_n\}$, the data clustering problem requires finding an optimal partition $U = \{u_1, u_2, u_3, ..., u_k\}$ of X, where u_i is the i^{th} cluster/group of U, such that an overall similarity measure between the vectors that belong to the same group, or an overall dissimilarity measure of the vectors that belong to different groups, is maximized, in terms of a given cost function $f(U)$.

After applying any clustering algorithm to a given set of data items, the quality of the resulting clusters must be evaluated in order to analyze the result in terms of objective measures [Halkidi et al., 2001]. There are different supervised and unsupervised measures that are used to evaluate the clustering results, including the Sum of Quadratic Errors (SSE) [Agustın-Blas et al., 2012], the Silhouette Width (SW) [Rousseeuw, 1987], the Davies-Bouldin Index (DB) [Davies and Bouldin, 1979] and the Rand Index (R) [Rand, 1971], among others. The most well-known evaluation method in the data clustering literature is based on the Euclidean distance which is used to give an overall measure of the error of the clustering. In this study, a well-known unsupervised distance measure known as the "sum of quadratic errors" (SSE) is adopted. Assuming that each data item has p properties, the SSE calculates a centroid vector for each cluster. The resulting centroid is also composed of p values, one for each property. The sum of the distances; i.e. errors; of each item's properties corresponds to the distances to the property values of the centroid of the cluster to which the item belongs.

$$error = \sum_{l=1}^{k} \sum_{i=1}^{n} W_{il} \sum_{j=1}^{p} (x_{ij} - u_{lj})^2 \tag{3.28}$$

In this equation:

$$\begin{aligned} k &\equiv \textit{the number of clusters}, \\ n &\equiv \textit{the number of items}, \\ p &\equiv \textit{the number of properties}, \\ W_{il} &= 1 \textit{ if the } i^{th} \textit{ item is in } k^{th} \textit{ cluster, and } 0 \textit{ otherwise}, \\ x_{ij} &\equiv \textit{the } i^{th} \textit{ item's } j^{th} \textit{ property, and} \\ u_{lj} &\equiv \textit{the center of the } j^{th} \textit{ property of the } k^{th} \textit{ cluster.} \end{aligned}$$

Also in this research, another well-known clustering validation technique known as the the Silhouette Width (SW) [Rousseeuw, 1987] is used. This measure allows the evaluation of an assignment of a given item to a given cluster, as well as the quality of a whole given solution. More importantly, this measure is commonly used for model selection, i.e. the selection of the best partitioning when different partitionings with different numbers of groups are given as valid solutions to the same clustering problem. This is carried out by firstly computing the Silhouette Width of the clustering solution for each k in a given range; then, the global maximum is identified and considered the estimated best solution to the given problem.

The Silhouette value (SV) of an individual item x in a given cluster indicates how good is the assignment of this particular item to this particular cluster. This is calculated as shown in Eq. (3.29) below.

$$SV(x) = \frac{b_x - a_x}{max(a_x, b_x)} \tag{3.29}$$

where a_x is the average distance from point x to to the other points in its own group, and b_x is the minimum average distance between point x and other points in other clusters. If there is only one item in a given cluster, the value of $SV(x)$ for that item is set to 0. This choice is made based on the recommendation given in Rousseeuw [1987]. Indeed, it can be seen from Eq. (3.29) that

$$-1 \leq SV(x) \leq 1 \tag{3.30}$$

where a Silhouette value of +1 indicates that the point is very distant from other clusters, and −1 indicates otherwise. For a single group, u_i, the Silhouette Width is given by Eq. (3.31).

$$SW(u_i) = \sum_{x \in u_i} SV(x) \tag{3.31}$$

For a given grouping solution, U_g, the Silhouette Width is calculated as the average Silhouette value of all data items, as given in Eq. (3.32).

$$SW(U_g) = \frac{1}{k} \sum_{i=1}^{k} SW(u_i) \tag{3.32}$$

Categorization of Clustering Algorithms

In the scientific literature, many clustering approaches have been proposed with different quality and complexity trade-offs. Each clustering algorithm is tailored to work on a specific domain space with no guarantee of finding optimal solutions for different data sets of different properties, sizes, structures, and distributions [Kashef and Kamel, 2010]. Traditionally, these approaches are classified to partitional, hierarchical and density-based approaches [Duda et al., 2001, Jain and Dubes, 1988].

[Handl and Knowles, 2007] and [Chang et al., 2009] categorize the clustering approaches into 3 and 4 groups, respectively, based on the clustering criterion being optimized. The first group adopted in both studies consists of the clustering algorithms that look for compact clusters by optimizing intra-cluster variations, such as variations between items that belong to the same cluster or between the items and cluster representatives. Such clustering algorithms tend to work well with well-separated clusters, but they often lack robustness with more complicated cluster structures. Well-known algorithms such as the k-means [MacQueen et al., 1967], model-based clustering [Dempster et al., 1977], average link agglomerative clustering [Voorhees, 1985] and self-organizing maps [Kohonen, 1990] belong to this category. The second group, according to both studies, consists of the clustering algorithms that strive for connected clusters by grouping neighboring data into the same cluster. Algorithms of this group are able to detect arbitrary-shaped clusters, but they are less effective when the spatial separation between clusters is small. Classical clustering techniques such as single link agglomerative clustering [Voorhees, 1985] and density-based methods [Ankerst et al., 1999] belong to this group.

The third group according to [Handl and Knowles, 2007] consists of clustering algorithms that look for spatially-separated clusters. However, this objective on its own may result

in clustering the oultiers individually while merging the rest of the data items into one big cluster. Clustering objective such as the Dunn Index and the Davies-Bouldin Index [Halkidi et al., 2001] combine this objective with other clustering objectives, such as compactness and connectedness, in order to improve the resulting solution. In contrast to the first two groups, this objective has not been used in any specialized clustering algorithm. According to [Chang et al., 2009], the third group includes simultaneous row-column clustering techniques known as bi-clustering algorithms [Madeira and Oliveira, 2004]. Finally, the fhe fourth group according to [Chang et al., 2009] includes the multi-objective clustering algorithms that seek to optimize different characteristics of the given data set [Mitra and Banka, 2006] along with the clustering ensembles approaches [Hong et al., 2008].

Evolutionary Clustering Algorithms

Different types of evolutionary algorithms (EAs) have been applied to clustering problems, either directly, or to improve existing clustering algorithms. In Krishna and Murty (1999), EAs are used to improve the K-means approach, producing the well-known genetic K-means algorithm, which outperforms the traditional K-means in hard clustering problems. Other works dealing with evolutionary K-means algorithms include [Xiao et al., 2010, Zahraie and Roozbahani, 2011].

Direct applications of EAs to clustering problems include [Hruschka and Ebecken, 2003], in which the objective function maximizes both the homogeneity within each cluster and the heterogeneity among clusters. This algorithm uses a simple encoding scheme that yields constant-length chromosomes. Deng et al. [2010] presented a genetic clustering algorithm that is based on a measure of mutual information between items of the different clusters. A real-encoding evolutionary algorithm with a special mutation operator and a fitness function based on the Davis-Bouldin index [Davies and Bouldin, 1979] is presented in [Liu et al., 2011]. This algorithm performed well in different synthetic problems and in the well-known Breast Cancer data set.

To the best of our knowledge, the only study that applied the grouping genetic algorithm [Falkenauer, 1998] to clustering problems is presented in [Agustın-Blas et al., 2012], in which the performance of a new grouping genetic algorithm that includes the traditional encoding of Falkenauer's grouping algorithms is tested for data clustering. The results were found to be very promising.

3.4 Summary

Following the emergence of the Genetic Algorithms in the seventies of last century, many different solution representations and search operators/heuristics were suggested and employed for grouping problems, with varying degrees of success. The main drawback of most of the suggested representations is that they fail to capture the essence of grouping problems which is to create groups that are meaningful with regard to a given problem domain [Falkenauer, 1996, 1998]. In this chapter, a literature survey of grouping problems and well-known representations is presented. Some of these representations suffer from high degrees of redundancy, i.e. multiple points in the search space are allowed to represent the same configuration of groups. Consequently, huge search space are created, which require longer periods of time to be explored. Additionally, based on previous studies, most of the suggested search operators/heuristics were found to either be context-insensitive or require high cost in terms of computation time when employed on grouping problems. In order to make conclusions about the effect of redundancy on the performance of grouping algorithms, a brief comparison between the performance of three different representations on a set of 15 problem instances from 3 different grouping domains is provided in chapter 5. This following chapter presents a detailed description of the proposed single point-based selection grouping hyper-heuristic framework.

Chapter 4

Single Point-Based Selection Hyper-heuristics for Grouping Problems

There are many studies in the literature that provide guidelines for designing hyper-heuristic frameworks for single-objective optimization. Burke et al. [2003] and Soubeiga [2003] provide general guidelines for designing effective single-objective optimization hyper-heuristic frameworks. On the other hand, studies on multi-objective optimization hyper-heuristic frameworks are scarce. Burke et al. [2003] presented a framework for hyper-heuristic for multi-objective combinatorial problems. Maashi et al. [2014] discussed the design issues related to the development of hyper-heuristics for multi-objective optimization, and proposed an online learning selection hyper-heuristic based on choice function for multi-objective optimization. However, all of these previous studies deal specifically with continuous optimization problems.

In this thesis, grouping problems are formulated as bi-objective discrete optimization problems. One of the two objective to be optimized in a grouping problem is the number of groups, k, which can only take discrete values. The other objective represents a problem specific cost/penalty/fitness function, that can take continuous values. Ideally, these two objectives should be optimized simultaneously. However, these two objectives are conflicting; i.e. each one of them can only be optimized at the expense of the other. In such cases, there might not be a single optimal solution. Instead, there could be multiple solutions with a trade-off from which a decision maker can choose. Those solutions are identified using the concept of *dominance* as illustrated in Figure 3.1 (page 29). A solution s_i is considered to dominate another solution s_j, ($s_i \succ s_j$) if, and only if, s_i is better than s_j in at least one objective, and s_i is not worse than s_j in any of the

other objectives. The set of the non-dominated solutions is known as the *Pareto optimal set*, and its image in the objectives domain is known as the *Pareto optimal front* [Zitzler and Thiele, 1998].

This chapter provides the description of a general selection hyper-heuristic framework based on single point-based search that is specifically designed to solve grouping problems. Inspired from the multi-objective optimization concepts above, this hyper-heuristic framework keeps track of a set of non-dominated solutions throughout the search process. Yet, the proposed approach is only designed for grouping problems, and cannot be used for other multi-objective optimization problems.

In addition to the common domain barrier found in traditional selection hyper-heuristics, the framework described in this chapter adds a further level of generality by embedding a solution representation barrier at which all different types of grouping problems are encoded using a particular grouping representation. Consequently, all grouping problems are treated similarly by the hyper-heuristic framework. The description of the framework given in this chapter includes a set of generic search operators/heuristics, that can be implemented with any grouping representation. Three different types of heuristics are described, which are merge, divide and change. Only the definition of the cost function along with the problem instances need to be fixed when applying the framework to a specific grouping problem. The solutions representation along with the low level heuristics (LLHs) remain the same across all domains. Figure 4.1 shows the layout of the framework.

The next few sections provide more details of the different components of the framework and their interaction with each other during the search process. Section 4.1 discusses how the initial set of non-dominated solutions is constructed. Section 4.2 describes a set of generic search operators/heuristics that can be used for different grouping problems. In section 4.3, a brief description of how a delta evaluation method is used by the framework to shorten the cost evaluation time. Sections 4.4 and 4.5 provide a description of the main iteration algorithm and the special two-phased acceptance algorithm of the framework, respectively. Finally, section 4.6 presents a brief summary of the work presented in this chapter.

4.1 Creating Initial Solutions

Typically, a single point-based selection hyper-heuristic framework starts the search process from one initial solution, and at each decision step throughout the search process,

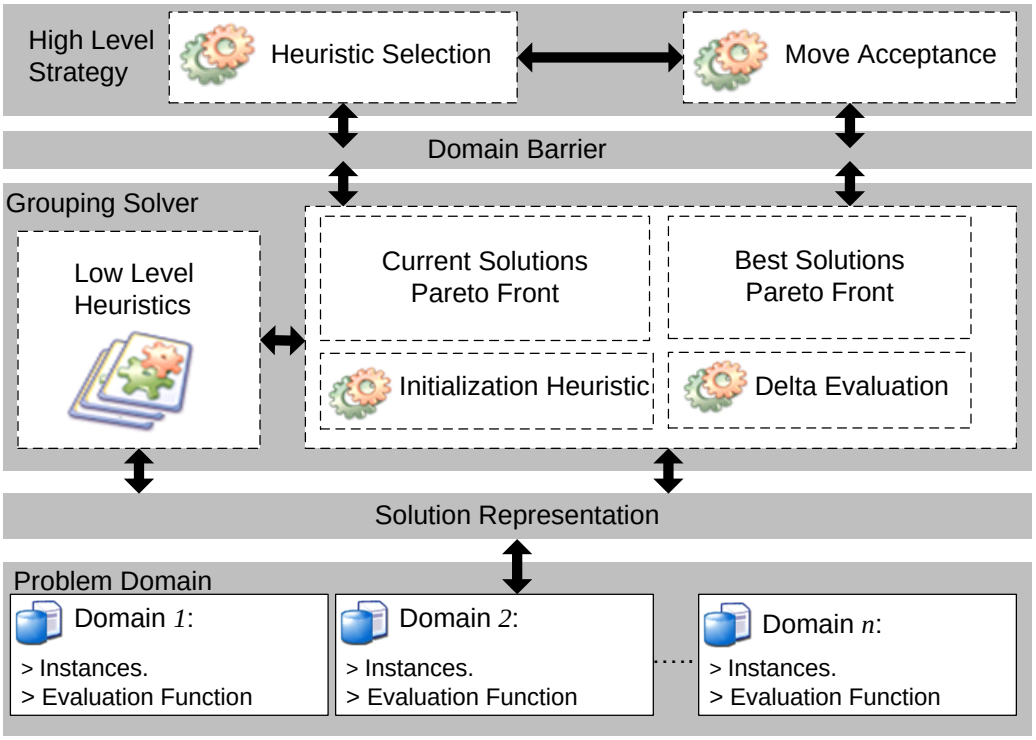

FIGURE 4.1: A selection hyper-heuristic framework for solving grouping problems.

it deals with a single solution to the problem being solved. On the other hand, multi-objective optimization frameworks, which are mostly population-based, commence by creating a population of solutions that are different from one another in terms of their quality with respect to each one of the objectives, and deal with all or a subset of that population at each decision point during the search [Coello et al., 2007, Zitzler and Thiele, 1998]. The hyper-heuristic framework described in this chapter attempts to exploit the bi-objective nature of the grouping problems, while adopting a single point-based search approach in order to capture the advantages of the two worlds. Similar to the multi-objective optimization frameworks, the grouping framework starts with and keeps track of a "set" of non-dominated solutions throughout the search. Also, similar to the typical single point selection hyper-heuristic frameworks, the grouping framework deals with only one point at any given decision step. Hence, the proposed framework is appropriate only for grouping problems not for general multi-objective optimization, which lies beyond the scope of this study.

Firstly, a random solution is created for each value of k in a given range $[LB, UB]$ in order to have a set of initial solutions with different number of groupings. Since the aim is to minimise the number of groups in a solution, choosing a large upper bound and/or a small lower bound for the number of groups will expand the search space unnecessarily. This, in turn, will slow the search process by causing the algorithm to explore unnecessary areas of the search space. For instance, in a grouping problem with

300 items that can optimally be distributed into 40 groups, an upper bound of 300, that is the $|number\ of\ items|$, will cause the algorithm to explore solutions that are very far from the promising ones. This range is arbitrarily decided in our experiments to align with previous work by Ülker et al. [2006]. Depending on the particular grouping problem being solved, reasonable upper and lower bounds can automatically be found by using problem-specific knowledge. For instance, for graph coloring problems, these bounds can be determined using algorithms such as the fast maximal clique approximation algorithm or by finding the maximal degree of the graph (the degree of the vertex with the maximum number of neighbors), as discussed in Ülker et al. [2006]. The exact and/or the approximate numbers of groups of all the test instances that are used in this study are already known. This knowledge is used to manually set the bounds for all the instances in this study.

Starting from the least value of k, i.e. from $k = LB$, one solution is created using an initialization heuristic. The initialization heuristic as a part of the problem domain implementation can be a "smart" problem-specific algorithm. However, in order to maintain a high level of generality, and to not give any bias to the framework and let it do the search, we have not used any such problem-specific initialization. On the other hand, we aimed to create a non-dominated initial set of solutions by using an initialization heuristic embedding a problem independent "smart" move appropriate for grouping problems. Starting from $k = LB$ through $k = UB$, a population of $(UB - LB + 1)$ non-dominated solutions are produced, where each solution is created for each integer value of $k \in [LB, UB]$.

The desired outcome of the initialization phase is a non-dominated set in which none of the solutions are dominated by any other solution. This dictates that, when comparing any two solutions, the one with the worse number of groups should have the better fitness value. However, since these solutions are created randomly, it is possible that a solution might violate this requirement. In order to enforce the dominance rule, the following check is performed during the initialization phase. For every newly created solution s_i at any $k = i$, apart from $i = LB$, the cost value is immediately computed and compared to the cost of the solution s_{i-1} at $k = i - 1$. Since the solution s_i is already worse in terms of the number of groups compared to the solution s_{i-1}, its cost value must be better; i.e. $f(s_i) < fs_{i-1}$. If this is not the case, that solution is discarded and re-created either randomly or by dividing one of the groups of the solution at $k = i - 1$. This way, the final outcome of the initialization phase is guaranteed to be a non-dominated set of solutions (See Algorithm 9).

Algorithm 9 *initialize*(): Constructing the Initial Non-Dominated Solutions Set

1: $Range \leftarrow [LB, UB]$ {for each value in the range, create a new solution}.
2: **for** $(i = LB \rightarrow UB)$ **do**
3: Create i empty groups. {For each group in the currently being created solution, randomly assign one item to each group, so that each group is guaranteed to have at least one item}.
4: **for** $(j = 1 \rightarrow i)$ **do**
5: $randomItem \leftarrow random[1, n]$ {select one item randomly}.
6: Insert the randomItem into the current group
7: **end for**
8: Randomly distribute the remaining items between the groups.
9: Compute the fitness value $f(s_i)$ of the resulting solution s_i.
10: **while** $(i \neq LB$ **and** $f(s_i) > f(s_{i-1}))$ (that is, s_i is worse than s_{i-1} in terms of the cost value as well as in the number of groupings) **do**
11: Discard s_i.
12: Create a new solution at i, either randomly by repeating this procedure again, or by applying a divide heuristic to the solution s_{i-1}.
13: **end while**
14: **end for**
15: Copy all created solutions to the current non-dominated set.
16: Copy the non-dominated set into an external archive that will keep track of the best non-dominated solutions.

4.2 A Set of Generic Heuristics for Grouping Problems

This section provides the description of a set of generic search operators/heuristics for grouping problems that can be implemented using a suitable solution representation. Based on previous findings in the literature such as in Falkenauer [1998] and Korkmaz [2010] as well as the results of the preliminary experiments that were performed in preparation for this study, the application of crossover operators on grouping problems was found to be disruptive and tends to impair the search rather than guide it. Additionally, since the proposed grouping framework performs a single-point based search, it has been decided that no crossover operators are to be implemented as part of this study. Instead, a set of effective generic operators that are able to transform the whole solution are implemented. Some of these operators are designed such that they are able to make large changes when applied on a given solution, which would reasonably diversify the exploration of the search space. On the other hand, some of these operators are smart operators which are enabled to make small modifications on a given solution leading to a potentially better/improved solution, intensifying the search process. Three different types of genetic operators were consequently developed, namely *merge*, *divide* and *change*. These three types were selected in order to cover the basic operations that can be applied to the groups; i.e, two or more groups can be *merged* into one group, a

group can be *divided* into two or more groups, and one or more items of one group can be *changed* to another group.

4.2.1 Merge Heuristics

The concept of this family of heuristics is to select two groups, u_i and u_j, from a selected solution and merge the objects of the two groups into one, u_{new}, as illustrated in Figure 4.2. Indeed, applying this operation results in *decreasing* the total number of grouping in the selected solution. The cost value of the new group u_{new} is either greater than or equal to the combined original cost values of u_i and u_j; i.e. $f(u_{new}) \geq f(u_i) + f(u_j)$. This is based on the fact that after the two groups are merged, the existing conflicts between the original members of each one of them will still be there. In addition to that, new conflicts might arise if any of the original members of one group has a conflict with any of the original members of the other group. This conforms with the expectation that, decreasing the number of groups lead to increasing the cost value. Three different versions of the merge heuristic were implemented in this study, which differ from each other in the way they select the groups to be merged in a given solution, as explained below.

- *Merge Random (M1)* heuristic selects the two groups randomly,
- *Merge Shortest (M2)* heuristic merges the two groups that contain the least number of items, and
- *Merge Least Conflicting (M3)* heuristic merges the two groups with the lowest partial dissimilarity values in the selected solution.

Indeed, merge heuristics produce relatively big jumps in the search space. Hence, merge heuristics can be considered as diversifying components.

4.2.2 Divide Heuristics

In a similar fashion, three divide heuristics are described as part of the grouping framework. Each one of these heuristics divides a selected group u_i into two groups, u_{new_1} and u_{new_2}, as illustrated in Figure 4.3. Applying a divide heuristic results in *increasing* the number of grouping in the selected solution. Also, some of the conflicting items in u_i may end up in different groups, which leads to the elimination of some conflicts. Consequently, the combined cost values of u_{new_1} and u_{new_2} is either less than or equal to the cost value of u_i; i.e. $f(u_{new_1}) + f(u_{new_2}) \leq f(u_i)$.

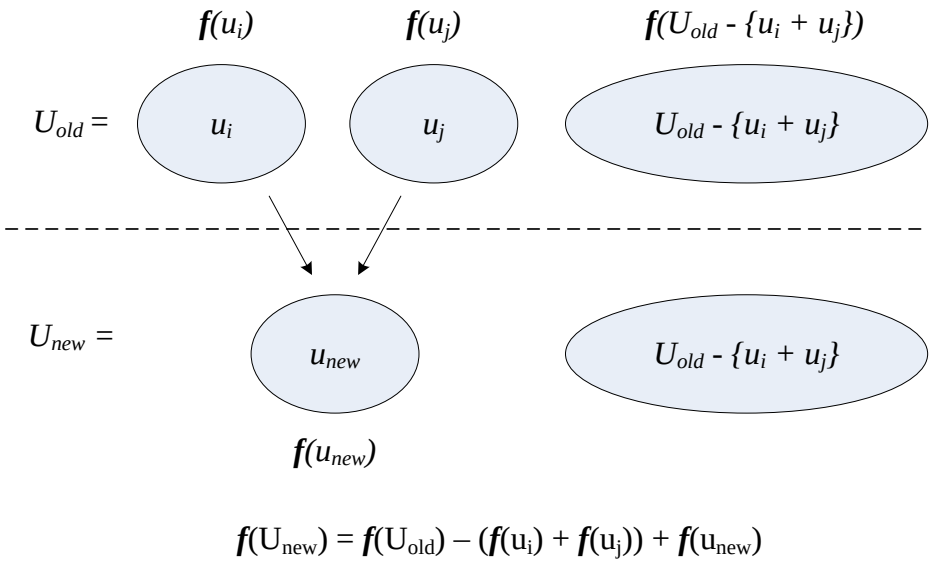

FIGURE 4.2: An example of a Merge type low-level heuristic.

- *Divide Random (D1)* heuristic selects the group to be divided completely randomly,
- *Divide Longest (D2)* heuristic divides the group that contains the largest number of items, and
- *Divide Most Conflicting (D3)* heuristic divides the group with the highest cost value in the selected solution.

Applying any of these heuristics result in increasing the number of the groups and, hopefully, decreasing the number of conflicts in the selected solution. This characteristic of divide heuristics is of particular interest as an intensification component; and it is used in the design of the local search algorithm that is used to improve the non-dominated set of solutions within the grouping framework, as explained in section 4.5.

4.2.3 Change Heuristics

Merge and divide heuristics produce relatively big jumps in the search space considering that they definitely influence the number of resultant groups for a given solution. However, the change heuristics attempt to make small modifications in a given solution by changing the group of a selected item while maintaining the same number of the groups after they are employed. This family of heuristics has four members.

- *Change Random (C1)* heuristic takes a random item x_m from a randomly selected group u_i and moves it into another randomly selected group u_j,

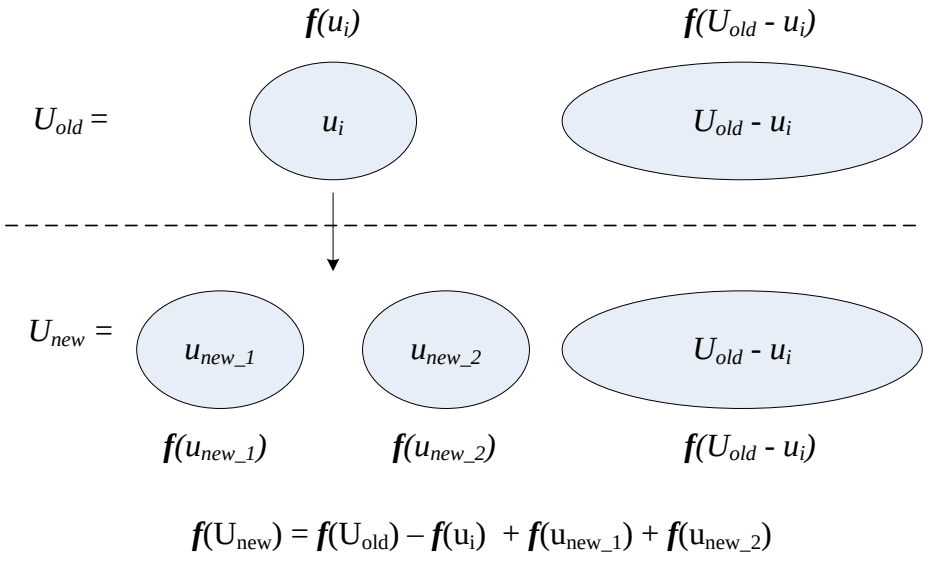

FIGURE 4.3: An example of a Divide type low-level heuristic.

- *Change Worst Item (C2)* heuristic finds the item that is causing the highest number of conflicts in a randomly selected group and moves it to another random group,
- *Change Worst Item from Most Conflicting Group (C3)* heuristic finds the group with the highest number of conflicts, and from that group it takes the item that is causing the highest number of conflicts and moves it to a randomly selected group, and
- *Change Worst Item from the Most Conflicting to the Least Conflicting Group (C4):* This heuristic is similar to $C3$, except that the item is moved to the group with the minimum number of conflicts rather than a random one.

4.3 Delta Evaluation

In the context of grouping problems as well as many similar combinatorial optimization problems, computing the cost of new solutions is the most time consuming part of the hyper-heuristic run. At each decision point, the solution resulting from the application of the selected LLH has to be evaluated before a decision about whether to accept or reject it can be made. Typically, this evaluation is carried out using the whole solution; i.e. calculating the partial cost of each group in the given solution. However, especially with large problem instances, this process is time consuming and greatly affects the solvers with time-based stopping criteria.

The generic search operators/heuristics implemented in this research allow the grouping framework to use a *delta evaluation* method, which requires computation of partial cost contributions of only the groups that have been affected by the application of the selected LLH. This would give a significant time advantage and consequently allow more iterations in each step. For example, if a change heuristic that moves a randomly selected item between two selected groups is applied on a solution U_{old}, and u_i and u_j are the groups that are involved in the process, the overall cost value $f(U_{new})$ of the resulting solution, U_{new}, is calculated using the equation below, where $u_i^{'}$ and $u_j^{'}$ are the resulting groups after the heuristic is applied:

$$f(U_{new}) = f(U_{old}) - (f(u_i) + f(u_j)) + (f(u_i^{'}) + f(u_j^{'})) \tag{4.1}$$

The illustrations given in Figures 4.2 and 4.3 show how the delta evaluation is used with the merge and the divide LLHs, respectively.

4.4 The Main Iteration of the Framework

The logical flow of the framework is presented in Algorithm 10. The first step is to create the initial non-dominated set of solutions using the *initialize*() algorithm (Algorithm 9) described above. Once the initialization phase is over, the framework proceeds to select one of the available LLHs using the hyper-heuristic selection method. The selected low-level heuristic is then applied on a randomly selected solution from the set of the non-dominated solutions created in the initialization phase. Although the framework keeps track of a set of solutions at any given decision point, the selected LLH is applied on only one solution at a time. In this regard, the behavior of the grouping hyper-heuristic framework is similar to the behavior of traditional single point-based hyper-heuristic frameworks. Hence, the single point notion of the grouping framework.

The number of groupings in the resulting solutions should always be kept within the boundaries of the $[LB, UB]$ range. Consequently, not all LLHs can be applied on all solutions. If the application of a LLH would increase the number of groupings in the selected solution, then that particular heuristic cannot be applied on the solution that contains the maximum allowed number of groups, that is the solution at $i = UB$. Similarly, a heuristic that decreases the number of groups in the selected solution cannot be applied on the solution at $i = LB$. If the number of grouping in the resulting solution falls outside the $[LB, UB]$ range, then the framework proceeds by simply jumping to the next iteration and selecting another solution and another LLH. Once the LLH is successfully applied, the fitness value of the resulting solution is evaluated (Algorithm

10, lines 3-9). Based on the implementation details of the grouping search operators, the fitness evaluation could be simplified using the delta evaluation method described in section 4.3 .

Algorithm 10 A selection hyper-heuristic algorithm for solving grouping problems

1: *initialize*() {See Algorithm 9}
2: **while** (elapsedTime < maxTime) **do**
3: Choose a random solution s_j from the current non-dominated set by $j \leftarrow UniformRandom(LB, UB)$.
4: Choose a LLH, LLH {Using the hyper-heuristic selection method}.
5: **if** ((LLH is *divide* **and** $j == UB$) **or** (LLH is *merge* **and** $j == LB$)) **then**
6: $CONTINUE$ {Select a different LLH and a different solution in order to ensure that the resulting solutions are kept within the $[LB, UB]$ range}.
7: **end if**
8: $s_{new} \leftarrow Apply(LLH, s_j)$ {s_{new} will have $i = (j-1)$ or j or $(j+1)$ number of groupings depending on the nature of LLH (merge or change or divide, respectively)}.
9: Compute $f(s_{new})$ (using delta evaluation, if the implementation permits).
10: $result \leftarrow moveAcceptance(s_{new}, s_i)$ {Use the hyper-heuristic acceptance method to compare the cost of s_{new} to the cost of s_i from the current non-dominated set}. {Following is the case when new solution is a worsening solution which is accepted by *moveAcceptance*}
11: **if** (($result$ is $ACCEPT$) **and** ($f(s_{new}) > f(s_i)$)) **then**
12: **if** ($f(s_{new}) > f(s_{i-1})$) **then**
13: Do nothing {s_{new} is rejected}
14: **else**
15: $s_i \leftarrow s_{new}$ {s_{new} is placed in the non-dominated set at grouping i, replacing s_i}.
16: **end if**
17: **end if**
18: **if** (($result$ is $ACCEPT$) **and** ($f(s_{new}) \leq f(s_i)$)) **then**
19: $s_i \leftarrow s_{new}$ {s_{new} is placed in the non-dominated set at grouping i, replacing s_i}.
20: $improveNonDominatedSet(i)$ {See Algorithm 11}.
21: **end if**
{if $result$ is $REJECT$ then do nothing and continue}
22: **end while**

4.5 Two-Phased Acceptance: Maintaining the Non-Dominated Sets During the Search

Maintaining a non-dominated set throughout the search requires that, when comparing any two solutions in that set, the solution with the worse number of groups should have the better cost value; and the solution with the better number of groups should have the worse cost value i.e. $f(U_i) > f(U_j), \forall\ i < j, (\forall i, j \in [LB, UB])$. These two

conditions are written explicitly because they are handled differently by the grouping hyper-heuristic framework, as will be explained shortly. During the search (Algorithm 10, lines 3-8), it is possible that a newly created solution might violate this requirement when compared to the other solutions in the non-dominated set. In other words, a new solution resulting from the application of any LLH might be either better or worse in *both* objectives when compared to some other solutions in the non-dominated set. The grouping hyper-heuristic framework handles such solutions by introducing a two-phased acceptance mechanism. The move acceptance of the hyper-heuristic constitutes the first phase of the acceptance, and does not make the final decision for the acceptance of a solution. This first phase can be considered as a pre-test component for the final acceptance. The new solution is only accepted after passing the multiple tests in the second phase of the framework acceptance mechanism.

In the first phase of the grouping framework acceptance mechanism, the hyper-heuristic move acceptance criteria compares the new solution s_{new} to the current solution s_i in order to make a decision regarding whether to *consider* the new solution for acceptance or reject it immediately (Algorithm 10, line 10). This is different from how the traditional hyper-heuristic frameworks operate, in which the decision made by the move acceptance is final. Some of the hyper-heuristic move acceptance methods that are used within this framework are non-deterministic, i.e. all improving solutions are considered for acceptance, while some of the worsening ones may or may not be considered.

In the second phase of the grouping framework acceptance mechanism, the framework differentiates between two main cases and the second case allows the use of local search to improve the non-dominated set of solutions further:

1. If s_{new} is considered for acceptance by phase one despite being worse than s_i in terms of cost value (Algorithm 10, line 11), then it is compared to s_{i-1}, such that the cost value of s_{i-1} acts as upper limit for the cost value of s_{new}. Hence, s_{new} is rejected if it is worse than s_{i-1}. Otherwise, it is accepted and inserted into the non-dominated set to replace s_i (Algorithm 10, lines 12-17).

2. If s_{new} is considered for acceptance by phase one and its cost value is better than, or equal to, the cost value of s_i (Algorithm 10, line 18), then it is accepted and inserted into the non-dominated set to replace s_i. A violation to the dominance rule may occur if s_{i+1}, which is already worse than s_i in terms of the number of groups, turns out to also be worse in terms of the cost value. In order to avoid this, s_{new} is then compared to s_{i+1} (Algorithm 11, line 2), which creates two cases:

    2.1 If s_{new} is worse than s_{i+1}, then there are no violations in the dominance rule, and no more action is needed (Algorithm 2, lines 2-3).

2.2 If s_{new} is better than s_{i+1}, then s_{i+1} is in violation of the dominance rule and consequently it is removed from the non-dominated set. A replacement solution is created by dividing a group in s_{new} using any of the divide heuristics (Algorithm 11, lines 4-8). The only remaining issue is that; this replacement solution at $i+1$ might have a better cost value than the solution at $i+2$, hence, the *for* loop in Algorithm 2. In the worst case scenario, this algorithm will be applied on all the solutions in the non-dominated set between i and UB. This process can be considered as a local search.

Algorithm 11 *improveNonDominatedSet*(i): Attempts to improve upon the cost of solutions in the non-dominated set starting from i^{th} solution to UB^{th} solution using a *divide* heuristic

```
1: for (j = i, UB) do
2:   if (f(s_(j+1)) ≤ f(s_j)) then
3:     BREAK {Further improvement is not possible}
4:   else
5:     Choose a random divide heuristic, LLDH
6:     s_new ← Apply(LLDH, s_j)
7:     s_(j+1) ← s_new {s_new is placed in the non-dominated set at grouping j, replacing
       s_(j+1)}.
8:   end if
9: end for
```

4.6 Summary

In this chapter, the description of a generic single point-based selection hyper-heuristic framework that is specifically designed for grouping problems is provided. Similar to multi-objective optimization methods, the framework keeps track of a set of non-dominated solutions throughout the search. Also, similar to single-objective optimization methods, the framework deals with only one solution at any given decision point during the search. Hence, the presented framework is only equipped to deal with grouping problems and is not suitable for general multi-objective optimization problems.

Selection hyper-heuristics involve a domain barrier that separates the hyper-heuristic strategy from the details of the problem domain. The main motive behind this barrier is to ensure that the hyper-heuristic strategy is kept as abstract as possible, so that it could be applied without modification on different problem domains with different sets of LLHs. In addition to this common barrier, the grouping hyper-heuristic framework presented in this chapter involves a representation barrier that adds a further level of generality by allowing the same set of LLHs to be used for different grouping domains. The aim is not to beat the state of the art techniques which are designed and tuned for

a particular problem domain. Rather, the aim is to provide the same selection hyper-heuristic framework based on a single point based search approach for all grouping problems.

This chapter also provided a description of a set of ten generic search operators/ heuristics that fall under three main categories. The merge type LLHs decrease the number of the groups in the solution on which they are employed, in contrast to the divide type LLHs which increase the number of groups. These two families of heuristics produce reasonably large jumps in the search space. The change LLHs move selected items between selected groups, and hence, they do not affect the number of the groups in the solution on which they are employed. These heuristics allow the grouping framework to use a delta evaluation method in order to reduce the cost evaluation time.

The presented framework also involves a special acceptance mechanism that ensures that the solutions in the non-dominated set do not dominate one another at any point during the search. This is achieved by a two-phased acceptance algorithm that uses the decision made by a traditional hyper-heuristic acceptance method only to decide whether to consider a solution for acceptance or to reject it at once. In the second phase, the solutions are subjected to multiple tests that may result in them being either rejected or used by a local search method to improve the current non-dominated set.

In the next chapter, a comparative analysis of the main features of the framework and associated implementation is provided. Firstly, the choice of the solution representation is scrutinized by comparing its performance against the performance of two other possible grouping representations. The purpose of this comparison is to make conclusions about the effect of the representation redundancy on the performance of the grouping operators. Additionally, the effect of the two-phased acceptance mechanism is analyzed by comparing the performance of the described grouping framework to a traditional single point-based selection hyper-heuristic framework.

Chapter 5

Preliminary Analysis of Grouping Hyper-heuristics Framework Components

The previous chapter presented a description of a single point-based selection hyper-heuristic framework embedding a set of search operators/heuristics that can be applied across grouping domains. This chapter provides a comparison between three implementations of this framework, based on three different solution representations, which are numeric encoding (NE), linear linkage encoding (LLE) and genetic grouping algorithm encoding (GGAE). Additionally, in order to analyze the effect of the two-phased acceptance algorithm that is part of the described grouping hyper-heuristic framework, this chapter provides a comparison between the performance of the grouping framework and the performance of a traditional hyper-heuristic approach.

5.1 Experimental Setup

This section provides the details of the criteria used to evaluate the performance of a given approach at the end of each experiment. It also presents the characteristics of the problem instances used during the experiments.

5.1.1 Evaluation Criteria

In this study, different hyper-heuristic selection and acceptance methods are used within the grouping framework described in the previous chapter. Each one of these approaches

starts from an initial set of non-dominated solutions and tries to find the best non-dominated set that can be attained in the given period of time for each run. The quality of the results obtained in each experiment is evaluated using the well-known hyper-volume metric [Zitzler and Thiele, 1998], which is used in evolutionary multi-objective optimization to evaluate the performance of search algorithms, by evaluating the spread of the solutions along the Pareto front, as well as the closeness of the solutions to the Pareto-optimal front. In Auger et al. [2009], the hyper-volume is defined as "a set measure used in evolutionary multi-objective optimization to evaluate the performance of search algorithms and to guide the search". This metric transforms multi-objective problems into single objective ones by comparing the hyper-volumes produced by each search algorithm at the end of the search. Hence, it provides a simple and effective method that can be used to compare the different hyper-heuristics used in this study.

Since the purpose of this study is to compare the performance of the grouping hyper-heuristic framework under different conditions, the hyper-volume metric is used to give an indication as to how far down the search space has the non-dominated set been pushed by each hyper-heuristic approach in each experiment. This is carried out by measuring the *size of the space covered (SSC)* by the final non-dominated solutions set obtained by each hyper-heuristic approach on each problem instance [Zitzler and Thiele, 1998] in each run. The further the non-dominated set is pushed down the search space, the better the corresponding approach. Comparing the final corner points obtained under different settings of the framework and the implemented search operators on each of the test instances is not considered here, since it is not needed to find the answers to the particular questions under investigation in this set of experiments.

5.1.2 Computing the Size of the Space Covered by a Non-dominated Set

Following the hyper-volume approximation presented in [Auger et al., 2009], the calculation of the hyper-volume value in this study, denoted as SSC, is approximated by the summation of the rectangular areas shown in Figure 5.1. The calculation of these rectangular areas was further simplified using the fact that, in the case of the grouping framework presented in this research, these rectangular areas have equal widths of one unit each. This is due to the fact that the number of groupings is incremental along the x-axis. The cost value of a fixed initial solution produced for the smallest allowed number of coloring, $k = LB$, for a given instance is used as a reference to compute the hyper-volume of the final non-dominated solutions set obtained by each algorithm on that instance.

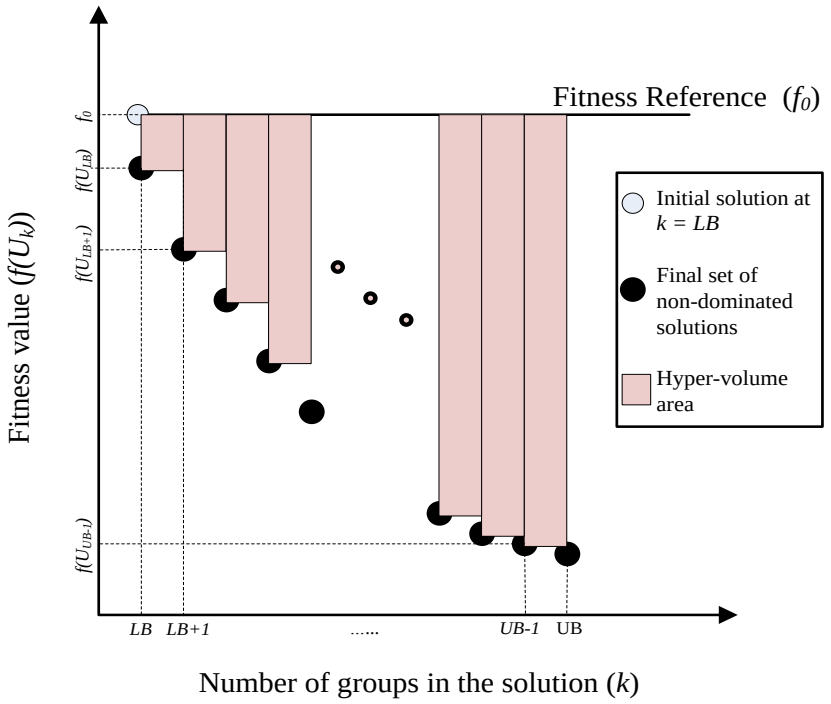

FIGURE 5.1: Approximating the calculation of the size of the search space covered by a final non-dominated solutions set using rectangular areas.

Mathematically, the final equation used to compute the hyper-volume value is derived as shown in the equations below. In this derivation, $SSC(i, i+1)$ is the size of one rectangular area between any two points that have consecutive values of k; and $SSC(LB, UB)$ is the total size of all rectangular areas between $k = LB$ and $k = UB$. The larger the hyper-volume, the larger the size of the space covered by the non-dominated set, and hence, the better the corresponding hyper-heuristic approach.

$$\textit{since} \qquad SSC(i, i+1) = ((i+1) - (i)) \times (f_0 - f(U_i)) \tag{5.1}$$

$$= f_0 - f(U_i) \tag{5.2}$$

$$\textit{then} \qquad SSC(LB, UB) = \sum_{i=LB}^{UB-1} f_0 - f(U_i) \tag{5.3}$$

$$= (UB - LB) \times f_0 + \sum_{i=LB}^{UB-1} f(U_i) \tag{5.4}$$

5.1.3 Experimental Data and Settings

The grouping hyper-heuristic framework and embedded search heuristics are designed to be generic and applicable across different grouping domains. In this set of experiments, 15 different benchmark instances from 3 different domains, namely graph coloring, timetabling and data clustering, are used. The characteristics of these instances are summarized in Table 5.1. For graph coloring, 5 benchmark instances from the well-known DIMACS challenge suite[1] are used, in which the number of colors, vertices, edges as well as edge densities vary. For examination timetabling, 5 instances from the Toronto benchmark suite referred to as "Toronto a" instances [Qu et al., 2009] are used. Similarly, 5 instances of data clustering are used, including the well-known breast-cancer, iris and the dermatology instances [Bache and Lichman, 2013] as well as 2 synthetic instances. Table 5.1 also shows the range of k values used during the experiments for each instance.

TABLE 5.1: The characteristics of the DIMACS, Toronto and data clustering problem instances used during the experiments. For DIMACS and Toronto, $|V|$ represents the number of vertices, $|E|$ the number of edges, % the edge density and $k^*/\chi(G)$ represent the best known number of colours (or time-slots) and the chromatic number respectively [Wu and Hao, 2012]. For data clustering, #items represents the number of items to be clustered and the #dimensions represents the number of attributes associated with each data item. L and U represent the lower and upper bounds for the k values used during the experiments.

		Graph Colouring				Range	
	Instance	$\|V\|$	$\|E\|$	%	$k^*/\chi(G)$	L	U
DIMACS	DSJC125.5	125	3891	0.50	17/?	13	23
	DSJC125.9	125	6961	0.89	44/?	40	50
	DSJC250.5	250	15668	0.50	28/?	24	34
	DSJC250.9	250	27897	0.90	72/?	68	78
	DSJC500.5	500	62624	0.50	48/?	43	53
		Examination Timetabling				Range	
	Instance	$\|V\|$	$\|E\|$	%	k^*	L	U
TORONTO	hec92	81	1363	0.42	17	12	22
	sta83	139	1381	0.14	13	8	19
	yor83	181	4691	0.29	19	13	25
	ute92	184	1430	0.08	10	6	15
	rye93	486	8872	0.08	21	16	27
		Data Clustering				Range	
	Instance	#items	#dimensions		k^*	L	U
Synth/Real	Iris	150	4		3	2	9
	Dermatology	366	34		6	3	10
	Synthetic1	450	2		3	2	9
	Synthetic2	500	2		5	2	9
	Breast-cancer	699	9		2	2	9

[1] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

In CHeSC 2011, the competing algorithms were compared using the median and the best objective values. Hence, the experiments were repeated 31 times each in order to allow finding the median easily. In this study, the competing algorithms are compared using the best value, and the median is not used. Hence, each experiment is repeated for 30 runs (trials), each of which is terminated when the best known coloring/timetable/-clustering is attained, or a time limit of 600 seconds is reached. All initial solutions are created randomly. Experiments were carried out on 3.6GHz Intel Core $i7-3820$ machines with 16.0GB of memory, running on Windows 7 operating system. Using the CHeSC 2011 benchmarking tool, this is equivalent to 1.4 times the amount of time suggested by the benchmarking tool on any other machine.

5.2 The Effect of the Representation Redundancy

As discussed in section 3.2, many solution representations that differ from each other in various aspects have been proposed in the scientific literature for grouping problems. Many of these representations suffer from high degrees of redundancy, i.e. they allow the same solution to be represented by many different points in the search space. Some researchers argue that, representation redundancy can be beneficial in some cases. For instance, Hancock [1992] has shown that, redundancy has the advantage of providing multiple configurations to solve the same problem. Hancock applied a genetic algorithm to learning neural network architectures, and concluded that, the redundancy allow the same hidden node configurations to be defined in a number of different ways. On the other hand, many researchers have argued that, redundant representations are not suitable for grouping problems [Falkenauer, 1996, Radcliffe and Surry, 1994, Tucker et al., 2005]. This is based on the observation that representing the same solution by more than one point in the search space would impair the search process, because the search algorithm will have to visit the same partitioning repeatedly.

In order to examine whether the representation redundancy actually affects the performance of the grouping hyper-heuristic approach, and to what extent if at all, the described grouping framework is implemented using three different solution representations from the scientific literature, namely numeric encoding (NE), linear linkage encoding (LLE) and genetic grouping algorithm encoding (GGAE) (See section 3.2). The NE is among the first solution representations that were explored in the literature. It is a straightforward representation, that represents the objects' membership by constant-length chromosomes, in which each gene corresponds to one object. Consequently, the value of each gene determines the group to which the corresponding object belongs. In section 3.2.1, it has been shown that, in the NE encoding, the same solution can

be represented by different chromosomes, and hence, the resulting search space is unnecessarily huge. In contrast, the LLE employs restrict conditions in order to get rid of the redundancy. According to Yılmaz and Korkmaz [2010], this is achieved at the expense of the computation time required by the search operators to maintain the LLE conditions during the search. Similarly, the GGAE is a redundant-free representation devised by Falkenauer [1998]. The main feature of the GGAE is that the associated search operators/heuristics are group-oriented, rather than object-oriented. In other words, the GGAE allows the operators to directly manipulate the basic building blocks of a grouping problem solution, which are the groups, rather than single objects which has almost no meaning when taken in isolation.

5.2.1 Experimental Design

The goal of this set of experiments is to examine the effects of the representation redundancy on the performance of the grouping hyper-heuristics. The main expected outcome of these experiments is that, the final non-dominated sets found by the operators that use redundancy-free representations, i.e the GGAE and the LLE representations, will be better than those found by the redundant representation, i.e. the NE representation. This is mainly due to the huge search space that the redundant representation has to explore. Also, it is expected that the results obtained using the GGAE will be better than those obtained using the LLE, due to the high computational cost required by the operators to maintain the LLE chromosomes during the search. Three implementations of the grouping low level search heuristics set described in section 4.2 were developed using the NE, LLE and the GGAE representations. Figures 5.2, 5.3 and 5.4 show examples of LLE Merge, Divide and Change low level heuristics (LLHs) respectively.

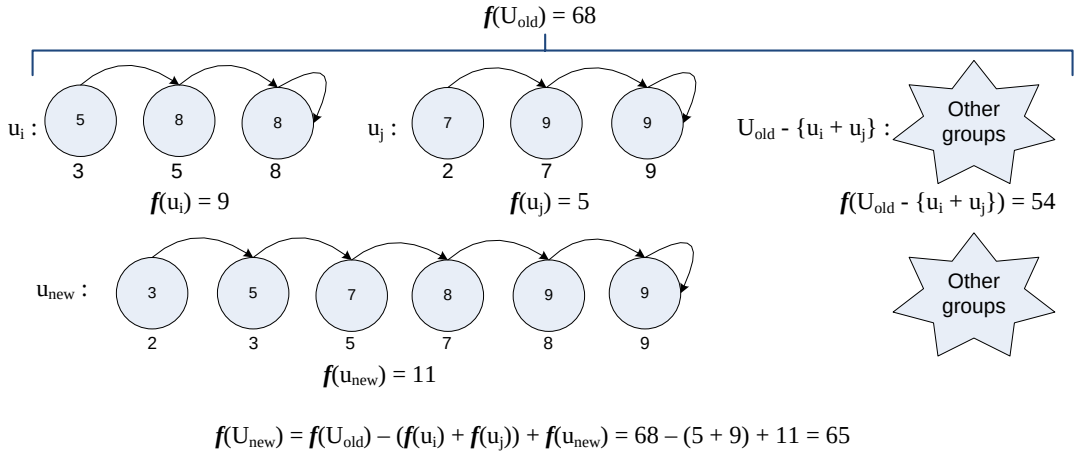

FIGURE 5.2: An example of a linear linkage encoding (LLE) Merge low-level heuristic

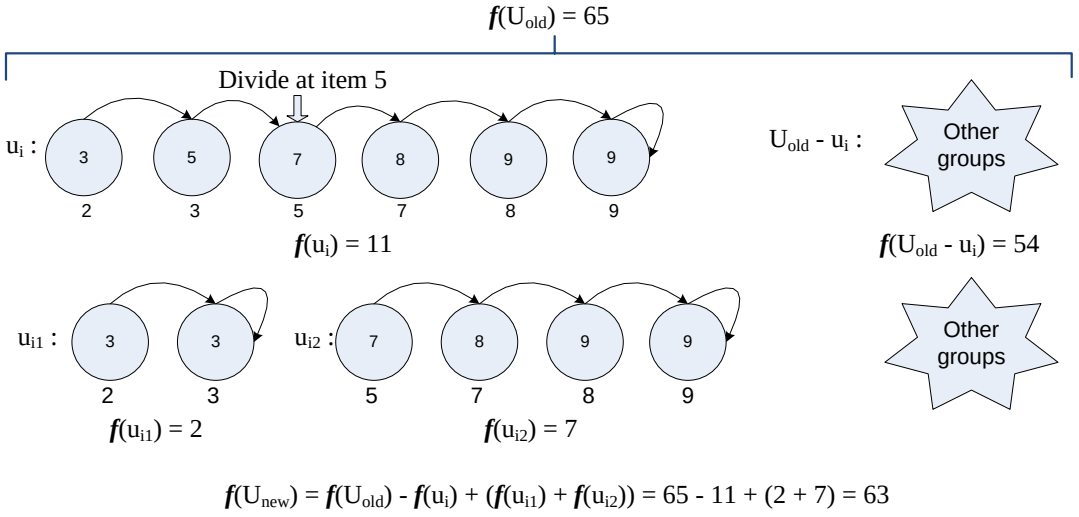

FIGURE 5.3: An example of a linear linkage encoding (LLE) Divide low-level heuristic

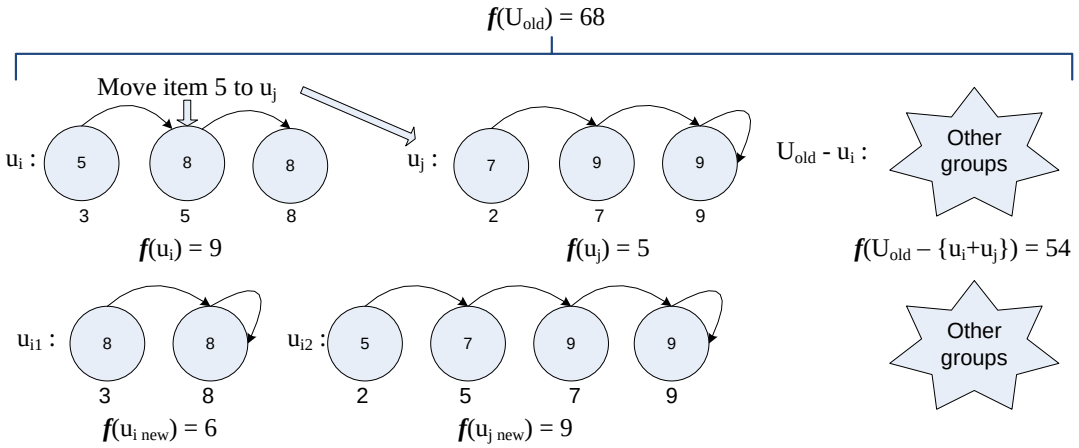

FIGURE 5.4: An example of a linear linkage encoding (LLE) Change low-level heuristic

Since the main focus of this study is not about how the LLHs are selected, nor about finding the best corner points, the choice of the hyper-heuristic selection and acceptance methods is kept simple, in order to give the representations greater effect on the results. The simple random (SR) method is used as the selection method of the hyper-heuristic. Also, the improving or equal (IEQ) method is used as the acceptance criteria. The grouping framework using one set of LLHs at a time was applied on the same problem instances under the same conditions. The final non-dominated sets produced by each approach at the end of each experiment were compared using the size of the space covered (the hyper-volume).

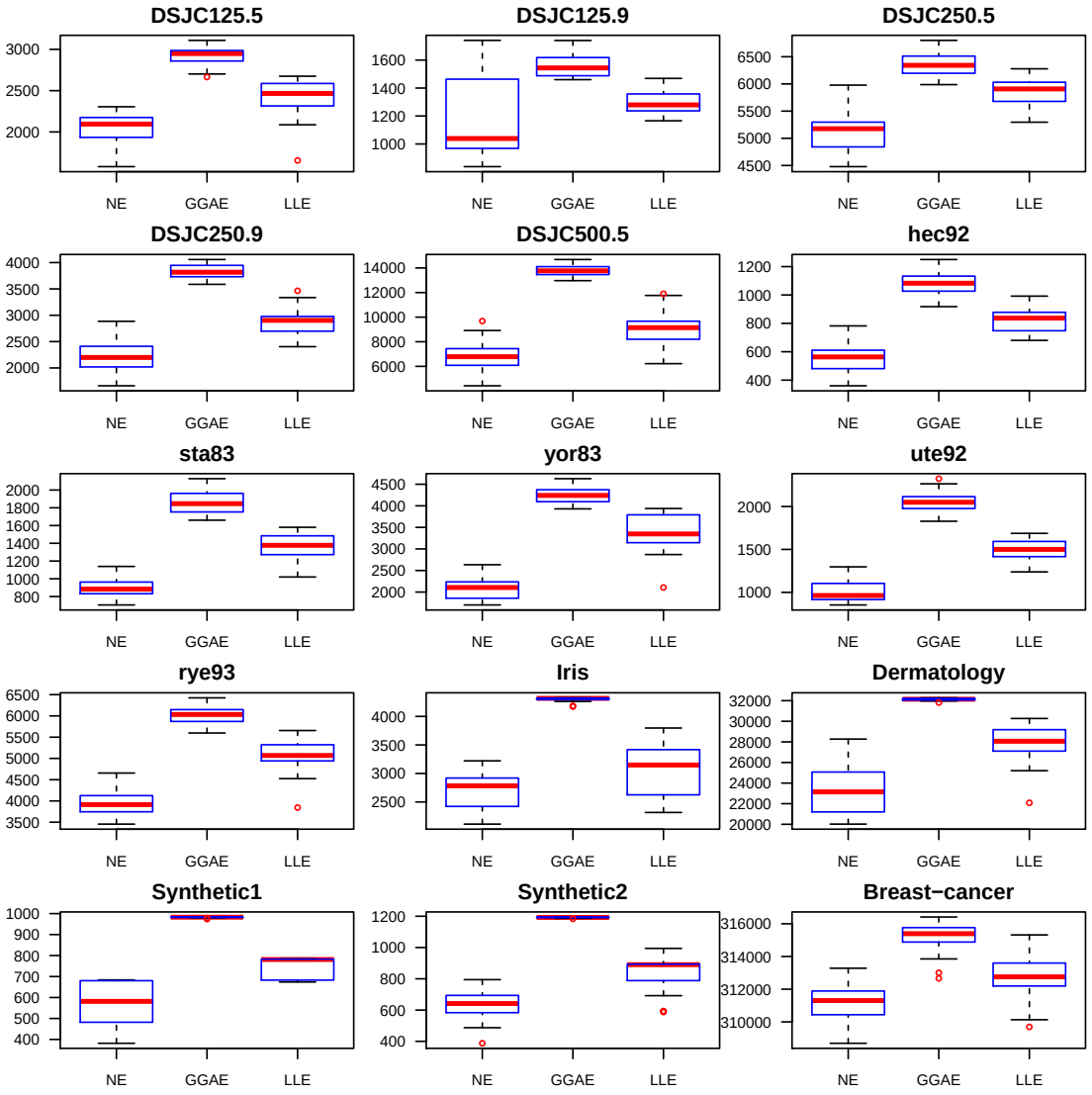

FIGURE 5.5: The effect of the representation redundancy: comparing the hyper-volumes obtained using three different implementations of the grouping hyper-heuristic framework. (NE) denotes numeric encoding, (GGAE) denotes grouping genetic algorithm representation, and (LLE) denotes linear linkage encoding.

5.2.2 Results and Discussion

The boxplots of the resulting hyper-volume values for each problem instance across the 30 runs are shown in Figure 5.5. In this figure, the results obtained using the numeric encoding are denoted as (NE), the results obtained using the genetic grouping algorithm encoding are denoted as (GGAE), and the results obtained using the linear linkage encoding are denoted as (LLE). The figure should be interpreted simply as follows: the higher the hyper-volume, the better the corresponding approach.

Indeed, it can clearly be seen that in all the instances that has been tested in this experiment, the hyper-volumes obtained using the redundancy free (GGAE and LLE)

are better than the those obtained using the redundant representation (NE).

In particular, the hyper-volumes obtained using the GGAE are always much higher than those obtained using the NE. With the exception DSJC125.9 and DSJC250.5 instances, even the worst hyper-volume values obtained by the GGAE are always much better than the best hyper-volume values obtained using the NE. Also, the results obtained using the LLE based implementation are better than the results obtained with the NE representation. In some cases however, the NE based implementation performs better than the LLE based implementation, but only in a few runs. In DSJC125.9 for instance, the result in some runs of the NE based implementation are better than their LLE counter parts.

When comparing the results of the implementations that use the redundant free representations, the GGAE and the LLE, the figures show clearly that the GGAE implementation results are always better than the LLE implementation results. The high cost in terms of computational time that is required to maintain the LLE definition during the search could be the main reason behind this observation.

The direct conclusion that is drawn from these results is that the GGAE representation is more suitable for the grouping problems compared to the LLE and the NE representations. This is due to the fact that the GGAE allows the search heuristics to be applied on the basic building blocks of grouping problems, which are the groups, while the LLE and the NE requires high cost to do the same thing. This conforms with the findings that Falkenauer [1996] and Tucker et al. [2005], among others, concluded in previous studies. However, the findings of this study do not provide enough evidence that can be used to safely generalise this conclusion to all redundant-free representations in comparison to redundant representations.

5.3 The Effect of The Two-phased Acceptance Mechanism

As has been discussed in section 4.5, the acceptance algorithm of the grouping hyper-heuristic framework employs a two-phased approach in order to maintain the dominance rule in the non-dominated set throughout the search. In the first phase, the framework makes a decision regarding whether to *consider* the solution resulting from applying the selected LLH on the current solution for acceptance, or reject it immediately. The framework uses a traditional hyper-heuristic acceptance method to make this preliminary decision. Clearly, this is a different strategy from the one that is traditionally found in selection hyper-heuristics, where the decision made by the hyper-heuristic acceptance method is final.

In the second phase, the framework attempts to maintain the dominance rule throughout the search process by comparing the solutions that successfully pass the first phase against some of the solutions that are already in the current non-dominated set at that decision point. Two tests are carried out in order to guarantee that the dominance rule is successfully maintained. Firstly, the capping test; which finalizes whether the new solution is accepted or not. The new solution is rejected if its cost value is worse than the cost value of the solutions with less number of groupings. Secondly, the propagation test which addresses the case where the new solution is better in terms of both objectives than some of the solutions that already exist in the non-dominated set; in which case, a local search method is applied to further improve the current non-dominated set. Sections 5.3.1 and 5.3.2 below analyze the effects of the components that are implemented into the second phase of the acceptance algorithm to ensure that the dominance rule is maintained throughout the search process.

5.3.1 Capping the Non-Deterministic Acceptance Methods

A solution with i groups, s_i, that successfully passes the first phase of the acceptance algorithm is firstly compared with the solution that has $i-1$ groups, s_{i-1}, in the current non-dominated set. Since s_i is already worse than s_{i-1} in terms of the number of groups, it has to be better in terms of the cost value in order for it to be accepted and inserted into the current non-dominated set, or else it will be rejected by the framework. In other words, the cost value of the solution s_{i-1} acts as an upper limit or a *cap* for, generally all the new solutions with any number of groups greater than $i-1$, but particularly for the solution with i number of groups. The fundamental reason behind introducing this cost value capping to the acceptance algorithm of the framework is to ensure that the dominance rule is maintained throughout the search. This is done despite of the decision made in phase one by the hyper-heuristic acceptance method which, on its own, could accept solutions with both worse number of groups and worse cost values compared to the solutions that are already in the current non-dominated set. In this section, the introduction of this upper limit is analyzed in order to study its effect on the performance of the whole framework.

Experimental Setup

The goal of this set of experiments is to study the effect of capping the cost values of the solutions that have successfully passed the first phase of the acceptance. This is carried out by comparing the quality of the results obtained using the two-phased acceptance

algorithm described in section 4.5 against the results obtained using a traditional hyper-heuristic acceptance scheme. Since the main focus of this study is not about how the LLHs are selected, the "simple random" (SR) method is used as the selection method of the hyper-heuristic. On the other hand, the "accept all" (ALL) method is used as the acceptance criteria since it guarantees that all worsening solutions will pass the first phase of the acceptance algorithm of the framework, and hence, will allow the second phase to have the greater effect.

Results and Discussion

A statistical summary of the results of the study is shown in Table 5.2. The left hand side of the table shows the results obtained using a traditional "Simple Random - Accept All" (SR-ALL) hyper-heuristic, whereas the right hand side of the table shows the results obtained using the capping concept in the second phase of the acceptance algorithm. For each of the 15 problem instance tested in this set of experiments, the maximum (max) and the minimum (min) hyper-volume values obtained during the 30 runs are shown, as well as the average value ($mean$) and the standard deviation (sd) of all the hyper-volume values obtained across the 30 runs. Also, Figure 5.6 demonstrates the boxplots of the hyper-volumes obtained across the 30 runs for each one of the 15 problem instance. In the figure, OFF represents the boxplots obtained using the traditional SR-ALL, whereas ON represents the hyper-volumes obtained using the capping concept in the second phase of the acceptance algorithm.

From the table, it can clearly be seen that all the maximum and average hyper-volume values obtained using the capping concept in the second phase of the acceptance algorithm are always greater than their counter parts that are obtained using the traditional SR-ALL hyper-heuristic approach. This fact is also clearly reflected in the boxplots of Figure 5.6, where all the plots denoted as ON are higher than those denoted as OFF, which indicates that the spaces covered by the final non-dominated sets are larger when the capping is ON than those obtained when the capping is OFF.

This observation becomes even clearer by looking at the results of the data clustering instances. In these 5 instances, even the *minimum* hyper-volume values obtained using the capping concept are greater than the *maximum* values obtained without capping. Again, this is reflected in the boxplots of Figure 5.6, where the whole box denoted as ON is higher than the whole box denoted as OFF, for all the data clustering instances.

The direct conclusion that can be drawn from this discussion is that overriding the decision made by the hyper-heuristic acceptance method using the cost value of the solution at i as an upper limit of the cost for the solutions that are considered for

acceptance at $i+1$ has a dramatic positive effect on the final results of the optimization process. In other words, regardless to the nature of the hyper-heuristic acceptance method used in the first phase and the decision it makes, maintaining the dominance rule throughout the search process by capping the cost values of the worsening solutions at $i+1$ using the cost values of the accepted solutions at i has a positive effect on the search process.

TABLE 5.2: Comparing the performance of a two-phased acceptance hyper-heuristic that uses the cost values capping concept to the performance of a traditional hyper-heuristic approach. *max* and *min* represent the maximum and minimum hyper volume values obtained throughout the 30 runs, and *mean* and *sd* represent the average and the standard deviation of all those hyper volume values.

		No Capping (No Upper Limit)				With Capping (With Upper Limit)			
	Instance	*max*	*min*	*mean*	*sd*	*max*	*min*	*mean*	*sd*
DIMACS	DSJC125.5	1935	1369	1634.13	138.72	1982	1436	1714.23	136.12
	DSJC125.9	962	579	762.37	97.63	1026	654	831.23	96.9
	DSJC250.5	3193	2259	2699	222.28	3386	2453	2888.83	213.57
	DSJC250.9	1737	1270	1460.5	118.22	1883	1402	1607.87	122.94
	DSJC500.5	5147	3610	4307.83	385.19	5643	3909	4765.07	401.40
TORONTO	hec92	939	604	769.37	84.00	973	643	802.5	84.23
	sta83	1606	1121	1325.47	118.60	1652	1191	1387.1	116.51
	yor83	2679	1966	2292.5	177.32	2770	2079	2385.77	177.40
	ute92	1686	1183	1437.6	113.25	1740	1272	1501.53	109.74
	rye93	3112	2294	2716	219.12	3309	2498	2905.77	226.18
Synth/Real	Iris	2318.44	1653.14	1902.06	134.50	3703.35	3413.14	3631.41	62.4
	Dermatology	3287.14	2434.58	2811.89	191.41	8390.79	4243.06	5594.93	938.99
	Synthetic1	83.27	61.41	73.65	4.36	230.99	143.82	184.56	22.73
	Synthetic2	105.77	79.17	93.08	5.60	319.10	226.01	258.71	23.45
	Breast-cancer	22434.14	17861.92	19670.62	1209.93	38066.78	28886.06	32145.12	2273.25

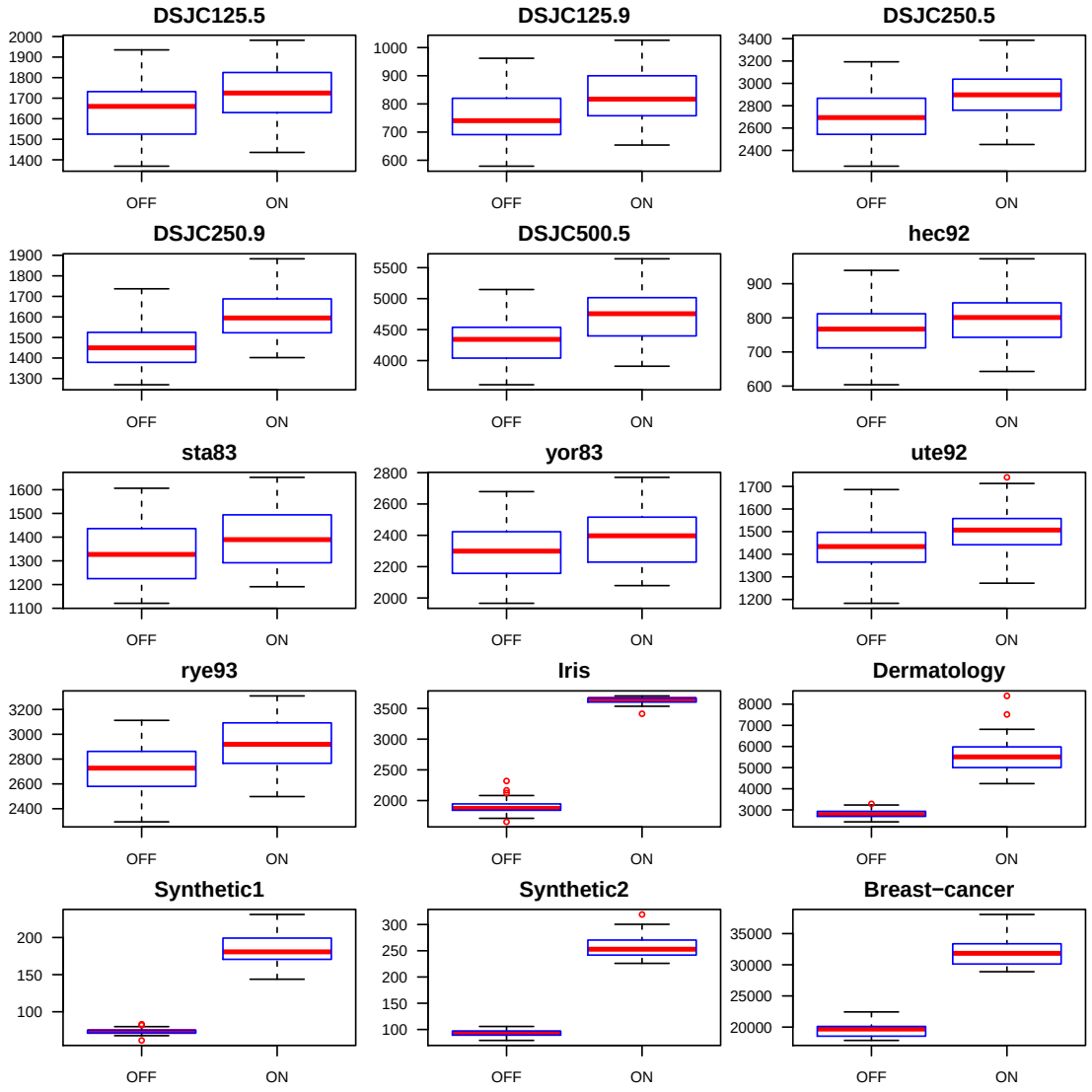

FIGURE 5.6: Maintaining the non-dominated set during the search: the effect of capping the cost values of the solutions that successfully passed the first phase of the acceptance process, by comparing the performance of the framework when the capping is used, denoted as (ON), to the performance of a traditional hyper-heuristic, denoted as (OFF).

5.3.2 Propagating the Improving Solutions

When a solution with i groups, s_i, successfully passes the first phase of the acceptance algorithm as well as the capping test of the second phase, it gets accepted and inserted into the current non-dominated set at the expense of the existing solution with i groups in the non-dominated set. The capping test guarantees that this newly accepted solution does not violate the dominance rule when compared to the solution with $i-1$ groups, as explained and discussed in the previous section. However, this newly accepted solution at i might still violate the dominance rule when compared with the solution at $i+1$.

Since s_i is already better than s_{i+1} in terms of the number of groups, the dominance rule is violated if the cost value of s_i is also better than the cost value of s_{i+1}. As discussed in the previous section, the cost value of the solution with the less number of groups should always act as the upper limit for the cost value of the solution with the greater number of groups.

The grouping framework addresses this situation by *propagating* the newly accepted solution at i when its cost value is better than the cost value of the existing solution at $i+1$. The propagation process involves the selection and application of a *divide* LLHs on the newly accepted solution. The pseudo code of the algorithm is shown in section 4.5. The following sections present and discuss the results and the effects of propagating the newly accepted improving solutions on the whole framework.

Experimental Setup and Data

The purpose of this study is to examine the effect of the propagation process. This is carried out by comparing the results obtained by enabling the framework to propagate the new solutions when they are better in terms of both the number of groups and the cost value against the results obtained when the propagation is disabled. Also, in order to focus the study on the effect of the propagation, the effect of the capping was eliminated by choosing a hyper-heuristic acceptance method that automatically eliminates the need for the capping. Namely, the "improving or equal" (IEQ) acceptance method is selected, which guarantees that all the solution that successfully passes the first phase will never need to be capped. Similar to the previous section, the "simple random" (SR) is used as the hyper-heuristic selection method. The evaluation criteria and the problem instances are the same as the ones used in the previous section. All experiments were terminated after 10 minutes.

Results and Discussion

The boxplots of the resulting hyper-volume values for each problem instance across the 30 runs are shown in Figure 5.7. The main observation is that the difference between the boxplots of the hyper-volume obtained with propagation (denoted as 'ON') and those obtained with the propagation (denoted as 'OFF') is small. In some of the DIMACS and the data clustering instances, the results obtained with propagation are slightly better than those obtained without propagation, whereas in the timetabling results there is almost no difference. The explanation to this observation lies in the fact that even when the propagation is enabled, it is not guaranteed to always happen. In other words, the rate of the occurrence of the case in which the propagation should happen cannot be

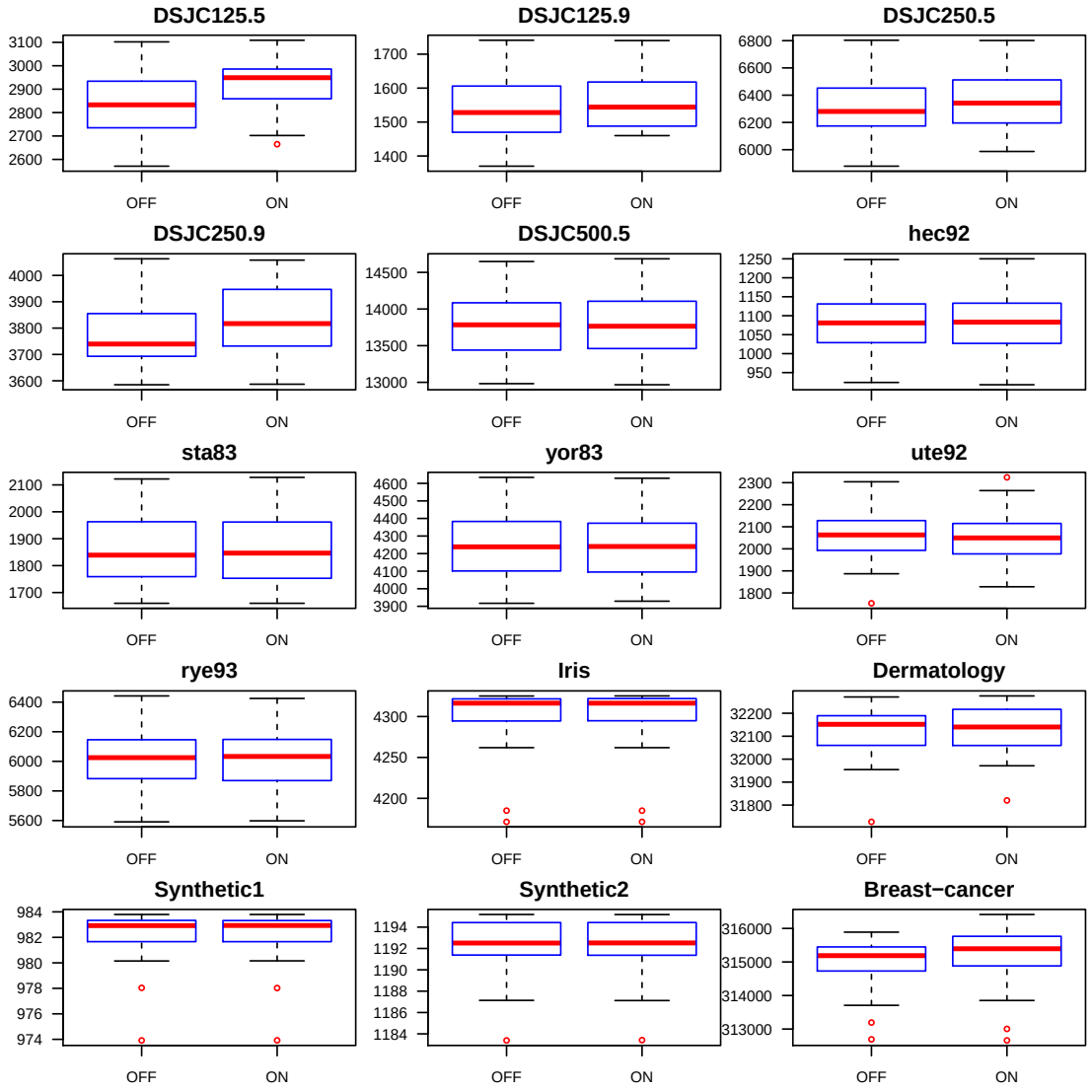

FIGURE 5.7: The effect of propagating the newly accepted solutions if they dominate other solutions in the current non-dominated set, by comparing the performance of the framework when the propagation is used, denoted as (ON), to the performance of a traditional hyper-heuristic, denoted as (OFF).

controlled. As a result, the propagation might happen in only a very small fraction of the total number of the iterations. Consequently, even if the propagation is enabled, it might not frequently get the chance to affect the acceptance decision. Bearing in mind that the capping is disabled (through the use of an IEQ acceptance criteria), this result tells us that the propagation, whenever it gets the chance to affect the decision process, can actually improve the final result.

Moreover, it is worth highlighting that the fundamental justification behind the introduction of the propagation concept is not only to have its direct effect on the decision process, but rather is to maintain the dominance rule and consequently co-operate with

the cost capping discussed in the previous section to achieve its purpose. Cost capping on its own, without the propagation, cannot work, i.e. cannot fully maintain the dominance rule throughout the search process.

5.4 Summary

In this chapter, a comparison between three different implementations of the grouping framework based on three different solution representations is provided. These are numeric encoding (NE), linear linkage encoding (LLE) and genetic grouping algorithm encoding (GGAE). The results show that, redundant-free representations (LLE and GGAE) perform much better than a redundant representation (NE). Also, the GGAE performs better than the LLE. Also, this chapter analyzed the effect of the two-phased acceptance algorithm that is part of the described grouping hyper-heuristic framework, by making a comparison between the performance of the grouping framework and the performance of a traditional hyper-heuristic approach. The results show that the capping test performed in the second phase of the acceptance mechanism of the grouping framework has a big positive effect on the the performance of the grouping framework. The propagation, on its own, has a small positive effect on the performance of the framework. However, the propagation is needed in order for the capping test to be meaningful. Based on these findings, the following chapter presents an extensive investigation of the performance of the grouping hyper-heuristic framework over a set of selected benchmark instances from graph coloring, examination timetabling and data clustering problem domains.

Chapter 6

Application of Grouping Hyper-heuristics on Graph coloring, Timetabling and Data Clustering

This chapter provides an investigation of the performance of the grouping hyper-heuristic framework over a set of selected benchmark instances from graph coloring, examination timetabling and data clustering problem domains. This investigation is carried out using different selection hyper-heuristics. These are formed using all the combinations of the 'simple random (SR)', the 'reinforcement learning (RL)' and the 'adaptive dynamic heuristic set (ADHS)' selection methods, denoted as {SR, RL, ADHS} respectively, with the 'late acceptance', the 'great deluge' and the 'iteration limited threshold accepting' move acceptance methods, denoted as {LACC, GDEL, ILTA} respectively;. A total of nine different selection hyper-heuristics is generated. From this point onward, a selection hyper-heuristic will be denoted as *heuristics selection-move acceptance*. For example, a hyper-heuristic that combines simple random selection method with great deluge move acceptance criterion will be denoted as SR-GDEL.

Although the ultimate goal of the hyper-heuristic research is to raise the level of generality, yet finding the position of the hyper-heuristics with respect to the state-of-the-art problem-specific methods is always of interest. Therefore, the performance of the approaches proposed based on the developed framework are further compared to previous approaches from the literature.

The empirical results show that a learning selection hyper-heuristic developed using the proposed framework turns out to be indeed sufficiently general and reusable. This hyper-heuristic either beats most of the previously proposed approaches tailored for the specific problem in hand or shows that it is highly competitive.

6.1 Experimental Design

Nine selection hyper-heuristics, each combining a heuristic selection method from {simple random (SR), reinforcement learning (RL), adaptive dynamic heuristic set (ADHS)} with a move acceptance from {late acceptance (LACC), great deluge (GDEL), iteration limited threshold accepting (ILTA)} are tested over three problem domains including graph coloring, examination timetabling and data clustering. The nine grouping hyper-heuristics, with the exact same settings without any modifications are applied to all of these three problem domains in order to examine the generality of the framework.

ADHS-ILTA is the state-of-the-art selection hyper-heuristic developed by Misir et al. [2013], which won the CHeSC 2011 competition across six hard computational problem domains. The remaining selection strategies and move acceptance criteria used in this study are instances from different categories. Simple random is a selection method with no learning; while reinforcement learning uses an online learning mechanism. The parameter employed within the late acceptance method is static (fixed); while the parameters of the great deluge criterion changed dynamically during the search process.

6.1.1 Setting the Parameters of the Hyper-heuristics Components

One of the main motives behind the hyper-heuristics is to produce systems that are general enough such that they can be applied to as many problems as possible with minimal expert intervention. One of the techniques to achieve this is to minimize the number of the parameters needed in both the heuristic selection and the move acceptance methods. The aim is to achieve parameter-free selection and acceptance methods, such as in the simple random selection method and the accept all solutions acceptance criterion. However, in reality, one can argue that although these so-called parameter-free methods appear to be free of parameters, their behavior actually depend on some pre-determined rules or values. The simple random, for example, is considered to be a parameter-free selection method. In fact, simple random selection method has one parameter which is the selection probability of each one of the available LLHs at each decision point during the search. The value of this single parameter is normally set beforehand such that each LLH is given an equal chance of being selected during the search. Consequently, the

task of *setting* that parameter during the search virtually disappears, but it still actually happens before the search starts. Misir et al. [2013] make similar arguments for all the other so-called parameter-free methods. As a result, it seems that developing a totally independent heuristic selection or move acceptance method is impossible. Alternatively, researchers seek to automate the process of setting the parameters and try to embed the capacity of the algorithm to control itself into it, in order to minimize the expert intervention, and to achieve the main goal of the hyper-heuristics, that is raising the level of generality. Following this argument, the parameter settings of the SR [Cowling et al., 2000], ADHS [Misir et al., 2013], and ILTA [Misir et al., 2013] followed the same settings as suggested in the literature. As for the reinforcement learning selection method, some initial experiments were conducted comparing different recommended settings from the literature. The final settings which are used in this study are as follows: the initial score value of all the LLHs is set to the same value that was calculated using the following equation:

$$initial\_score(h_i) = upper\ score\ pound - 2 * number\ of\ heuristics \tag{6.1}$$

The upper and lower bounds for the score of each one of the LLHs are set to 40 and 0, respectively, and the score increments and decrements values are both set to 1. The GDEL parameters shown in Eq. (2.3) are set to the following values: T is set to the maximum duration of a trial, ΔF is set to the minimum cost value in the initial non-dominated set and f_0 is set to 0 [Dueck, 1993]. A LACC approach that uses k separate lists of equal lengths, one for each value of $k \in [LB, UB]$, is adopted based on the results of some initial experiments. A separate list of 50 previous solutions is maintained for each one of the solutions in the current set of non-dominated solutions.

6.1.2 Trials Settings and CPU Specifications

Each experiment is repeated for 30 runs (trials). For the graph coloring and the timetabling problems, each run is terminated when the best known coloring/timetable is found, or when a time limit of 3600 seconds is exceeded. For the data clustering problems, the runs are terminated after a time limit of 600 seconds. For each one of the problem instances of all domains, 30 initial solutions are created randomly. In order to avoid initialization bias and to get fairly comparable results, all hyper-heuristic approaches operated on the same 30 initial solutions for each problem instance. Experiments were conducted on 3.6GHz Intel Core $i7-3820$ machines with 16.0GB of memory, running Windows 7 operating system.

6.2 Evaluation Criteria

The following subsections describe the details of the criteria used in this study to evaluate the performance of each approach in each experiment.

6.2.1 Comparing the Non-dominated Sets

Nine different hyper-heuristics were used in this study. Each one of them start from an initial set on non-dominated solutions and tries to find the best near-optimal non-dominated set that can be found in the given period of time for each run. In this study, the size of the space covered (SSC) by the final non-dominated solutions set obtained by each algorithm on each problem instance, also known as the *hyper-volume* metric [Zitzler and Thiele, 1998], is calculated. These hyper-volume values are used to compare the performances of different hyper-heuristic approaches against each other. The hyper-volume is a well known metric that is commonly used to evaluate the spread of the solutions along the Pareto front, as well as the closeness of the solutions to the Pareto-optimal front. Hyper-volume values are calculated as explained in section 5.1.1. The larger the hyper-volume, the larger the size of the space covered by the non-dominated set, and hence, the better the corresponding hyper-heuristic approach.

6.2.2 Finding the Best Corner Points

The proposed approach cannot be used for multi-objective optimization, but it operates based on the dominance concept from the multi-objective optimization. When searching for the minimum coloring in a graph coloring problem, the aim is not just to minimize the violations, but also to get rid of all violations, yielding 0 violations as the objective value. Still in our approach, a set of non-dominated solutions for a given range of number of colors/groupings is archived. Hence, in this study, the best corner point for a given hyper-heuristic on a given graph coloring or timetabling problem instance is considered to be the minimum number of colors achieved with no violations. For instance, Figure 6.1 shows the final Pareto-fronts obtained by the nine hyper-heuristics on three graph coloring instances in one of the 30 runs performed. For the DSJC250.1 problem instance, the figure shows that all the nine hyper-heuristics found the same minimum number of colorings that has no violations at $k = 9$. However, for the DSJC250.9 problem instance, only three hyper-heuristics found a minimum number of colorings that has no violations at $k = 73$

The same metric is used in the case of data clustering problems. However, since the cost function of the data clustering problem is calculated using the concept of the quadratic

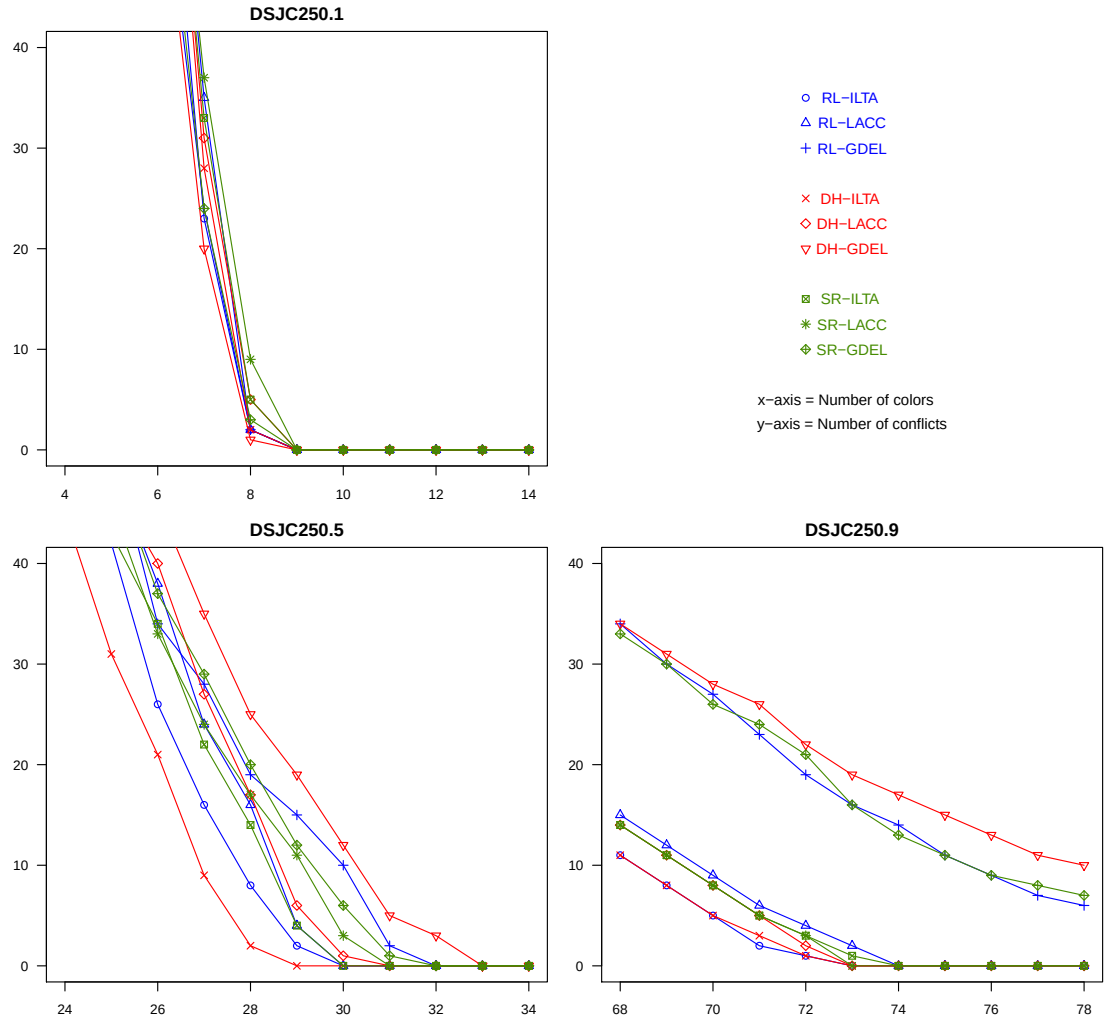

FIGURE 6.1: Deciding the best corner points from the final Pareto-fronts obtained by the hyper-heuristics on DSJC250.1, DSJC250.5 and DSJC250.9 problem instances.

distance, as explained in section 3.3.3, it is impossible for any solution to yield an overall cost value of 0 distance units, even for the best known solutions for the tested problem instances. Alternatively, the corner points are identified using the Silhouette Width (SW) metric explained in section 3.3.3, which is a well-known technique that is used in data clustering analysis to identify the best corner points in a given partitioning [Rousseeuw, 1987]. The Silhouette value (SV) of an individual item x in a given cluster indicates how good is the assignment of this particular item to this particular cluster.

6.2.3 Success Rates and Pairwise Statistical Comparisons

Another performance metric that is used in this study is the *success rate*, denoted as ($sRate\%$). This performance metric indicates the percentage of the runs in which the expected (best) number of groups/corner points has been successfully found by a given

hyper-heuristic approach. It is calculated by simply dividing the number of the trials in which a given hyper-heuristic approach found the expected best solution by the total number of trials, which is set to 30, and then multiply the result by 100. Similarly, the average time (in seconds) taken to achieve those success rates. This is calculated using the durations of the successful runs only.

$$sRate\%(HH_i(instance_j)) = \frac{\sum successful\_runs(HH_i(instance_j)) * 100}{total\ number\ of\ runs} \tag{6.2}$$

$$\mu(t, HH_i(instance_j)) = \frac{\sum t(successful\_runs(HH_i(instance_j)))}{number\ of\ successful\ runs} \tag{6.3}$$

Also, and in order to be able to make better comparisons between the performance of the different hyper-heuristics used in this study, the Wilcoxon Signed Rank test is used as a statistical test for the average pairwise performance comparison of algorithms. The statistical test was carried out based on the minimum coloring/grouping each hyper-heuristic managed to find over the 30 runs for each problem instance.

6.3 Experimental Data

The characteristics of the benchmark problem instances used during the experiments for the graph coloring and timetabling problems are summarized in Table 6.1 and 6.2. For the graph coloring problem, the nine hyper-heuristics are applied on 19 benchmark instances. These instances differ from each other in terms of number of colors, vertices, edges as well as edge densities. The instances in the upper half of Table 6.1 are taken from the COLOR02 website [1], which was initially compiled for the purposes of a graph coloring competition. The graphs called "Myciel" are based on the Mycielski transformation and are considered to be difficult to solve, and the coloring number increases in problem size. Also each one of the queen$n.n$ graphs is a graph on n^2 nodes, each of which represents a square on an $n \times n$ chessboard. If any two squares are in the same row, column, or diagonal, then their corresponding nodes on the graph are considered to be connected by an edge. The objective is to place n sets of n queens each on the board so that no two queens of the same set can capture one another. This could be achieved only if the graph has a coloring number n. In addition to these data sets, problem instances in the bottom half of the same table are taken from the well known DIMACS[2] challenge suite. For examination timetabling, instances from the Toronto benchmark suite referred to as

[1] http://mat.gsia.cmu.edu/COLOR02/

[2] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

Toronto a instances Qu et al. [2009] are used. Tables 6.1 and 6.2 also show the range of the k values used during the experiments for each one of the problem instances.

TABLE 6.1: The characteristics of the COLOR02 and DIMACS graph coloring problem instances used during the experiments. $|V|$ represents the number of vertices, $|E|$ the number of edges, % the edge density and $\chi(G)$ represent the best known chromatic numberWu and Hao [2012]. L and U represent the lower and upper bounds for the k values used during the experiments.

	Graph Colouring					Range	
	Instance	$\|V\|$	$\|E\|$	%	$k^*/\chi(G)$	L	U
COLOR02	myciel3	11	20	0.40	4/4	2	9
	myciel4	23	71	0.28	5/5	2	10
	myciel5	47	236	0.22	6/6	3	11
	queen5.5	25	160	0.53	5/5	2	10
	queen6.6	36	290	0.46	7/7	4	12
	queen7.7	49	476	0.40	7/7	2	12
	queen8.8	64	728	0.36	9/9	6	14
DIMACS	le450_25a	450	8260	0.08	25/25	20	30
	le450_25b	450	8263	0.08	25/25	20	30
	le450_25c	450	17343	0.17	25/25	20	30
	le450_25d	450	17425	0.17	25/25	20	30
	DSJC125.1	125	736	0.09	5/?	2	10
	DSJC125.5	125	3891	0.50	17/?	13	23
	DSJC125.9	125	6961	0.89	44/?	40	50
	DSJC250.1	250	3218	0.10	8/?	4	14
	DSJC250.5	250	15668	0.50	28/?	24	34
	DSJC250.9	250	27897	0.90	72/?	68	78
	DSJC500.1	500	12458	0.10	12/?	7	17
	DSJC500.5	500	62624	0.50	48/?	43	53

TABLE 6.2: The characteristics of the Toronto timetabling problem instances used during the experiments. $|V|$ represents the number of vertices, $|E|$ the number of edges, % the edge density and k^* represent the best known number of time-slotsWu and Hao [2012]. L and U represent the lower and upper bounds for the k values used during the experiments.

	Examination Timetabling					Range	
	Instance	$\|V\|$	$\|E\|$	%	k^*	L	U
TORONTO	hec92	81	1363	0.42	17	12	22
	sta83	139	1381	0.14	13	8	19
	yor83	181	4691	0.29	19	13	25
	ute92	184	1430	0.08	10	6	15
	ear83	190	4793	0.27	24	20	29
	tre92	261	6131	0.18	23	18	28
	lse91	381	4531	0.06	18	14	23
	kfu93	461	5893	0.06	20	16	24
	rye93	486	8872	0.08	21	16	27
	car92 I	543	20305	0.14	32	28	37
	uta92 I	622	24249	0.13	35	31	39
	car91 I	682	29814	0.13	35	31	39

TABLE 6.3: The characteristics of the synthetic, Gaussian and real-world data clustering problem instances used during the experiments. N represents the number of items, D the number of dimensions/attributes and k^* represents the best number of clusters [Handl and Knowles, 2007]. L and U represent the lower and upper bounds for the k values used during the experiments.

	Data Clustering				Range	
	Instance	N	D	k^*	L	U
Synthetic	Square1	1000	2	4	2	9
	Square4	1000	2	4	2	9
	Sizes5	1000	2	4	2	9
	Long1	1000	2	4	2	9
	Twenty	1000	2	20	16	24
	Fourty	1000	2	40	36	44
Gaussian	2D-4c	1623	2	4	2	9
	2D-10c	2525	2	10	6	14
	2D-20c	1517	2	20	16	24
	2D-40c	2563	2	40	36	44
	10D-4c	958	10	4	2	9
	10D-10c	3565	10	10	6	14
Real	Zoo	101	16	7	3	11
	Iris	150	4	3	2	7
	Dermatology	366	34	6	2	10
	Breast-cancer	699	9	2	2	7

For the data clustering problem, 16 problem instances were used in this study in order to examine the performance of the grouping hyper-heuristics with instances that have different properties and sizes. The first 12 of those instances are taken from Handl and Knowles [2007]. The characteristics of these instances are shown in Table 6.3 and Figure 6.2. In the figure, the top 6 instances are 2-D hand-crafted problem instances. The purpose of these instances is to study the ability of the framework to successfully identify clusters in instances that contain interesting data properties, such as different cluster sizes and high degrees of overlap between the clusters. Data instances Square1, Square4 and Sizes5 contain four clusters each. The main difference between these instances is that clusters in Square1 and Square4 are of equal size, whereas the clusters in Sizes5 are not. Clustering algorithms that rely on the cluster connectedness [Dempster et al., 1977, MacQueen et al., 1967] to evaluate the quality of the clusters have poor performance with these data sets due to the high overlap between the clusters. On the other hand, data instance Long1 consists of two well-connected long clusters. As a result, algorithms that use cluster connectedness to evaluate the clusters can easily identify these two clusters, unlike algorithms that use cluster compactness [Chang et al., 2009, Handl and Knowles, 2007]. Finally, problem instances Twenty and Forty exhibit a mixture of the properties discusses above, and hence, they are hard to solve for either types of algorithms, whether they are based on cluster compactness or cluster connectedness.

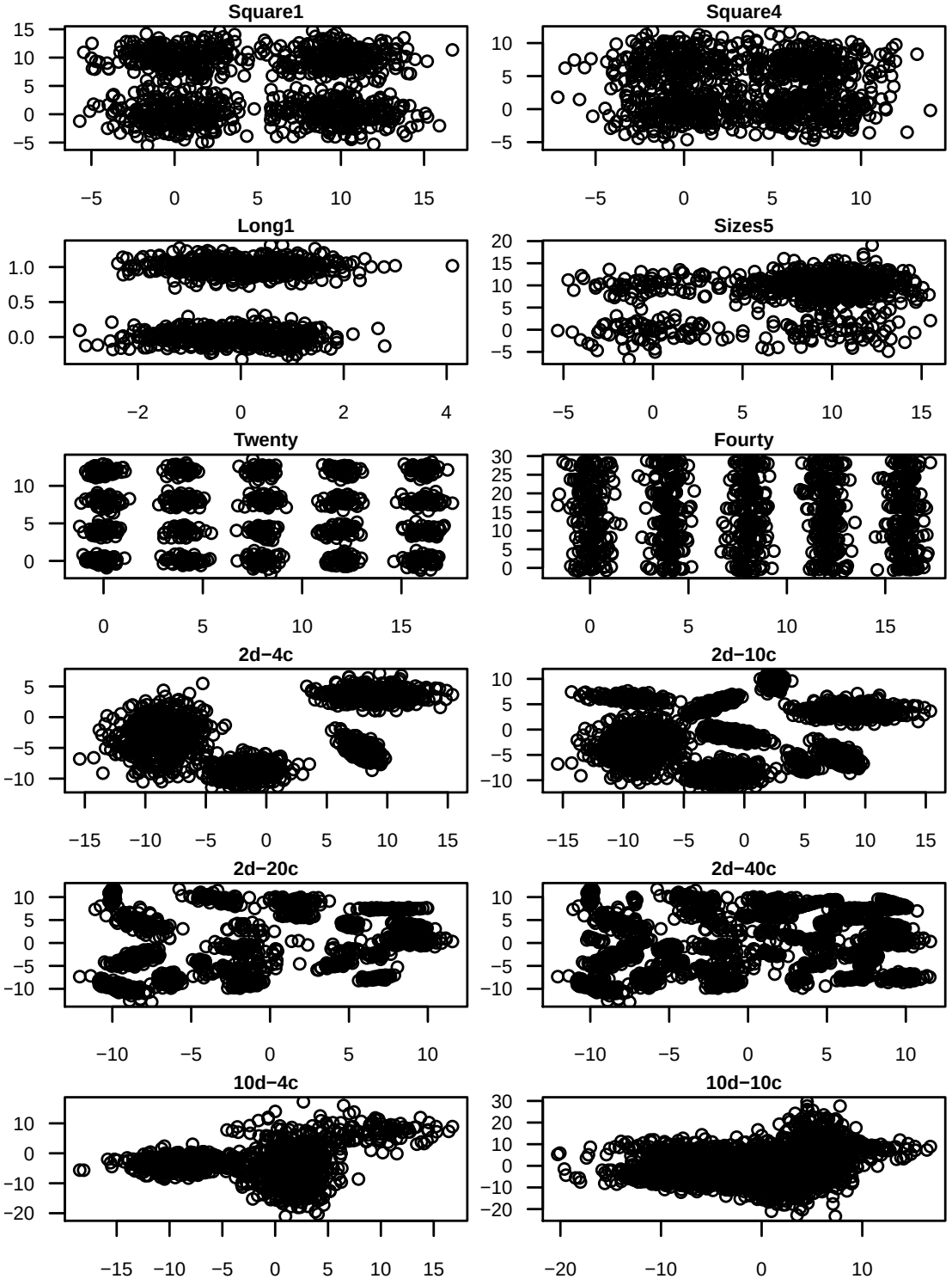

FIGURE 6.2: The synthetic data clustering problem instances used in the experiments. The top six instances are handcrafted data sets. Each one of Square1 and Square4 contain four clusters of equal size, whereas the four clusters in Sizes5 are not. These problems are hard to solve for algorithms that evaluate clusters using connectedness, due to the high overlap between the clusters. On the other hand, Long1 consists of two elongated clusters that can easily be found by algorithms that use connectedness, but not by those using compactness. The bottom six instances are randomly generated data sets. 2D-4c, 2D-10c and 2D-20c are 2-D instances containing 4, 10 and 20 Gaussian clusters, respectively. Similarly, 10d-4c and 10D-10c are 10-D instances (the figures show their 2-D projections only) containing 4 and 10 clusters, respectively.

The 6 problem instances on the bottom half of Figure 6.2 are randomly generated instances that were created using the Gaussian cluster generator described in Handl and Knowles [2007]. Instances 2D-4c, 2D-10c, 2D-20c and 2D-40c are all 2 dimensional instances containing 4, 10, 20 and 40 clusters, respectively. Similarly, instances 10D-4c and 10D-10c are 10 dimensional instances that contain 4 and 10 clusters, respectively. Additionally, 4 real world problem instances were used in this study. These are taken from the UCI Machine Learning Repository [Bache and Lichman, 2013], which currently maintains around 300 data sets as a service to the machine learning community.

These selected real-world instances differ from each other in many ways, such as in the number of clusters, number of dimensions as well as the data type of the values in each dimension. For example, Iris instance contains 3 equal-sized clusters of 50 data items each, and the data type of the values of each dimension is continuous, whereas Dermatology instance contains 6 clusters of different sizes of $(112, 61, 72, 49, 52, 20)$ data items, and the data type of each dimension is integer.

Also, within each instance, the actual values for the different dimensions were measured on different scales. Consequently, some data processing was necessary before applying the grouping hyper-heuristics on these clustering instances. In this pre-processing, the real data is normalized such the mean is equal to 0 and the standard deviation is equal to 1 in each dimension. Initial experiments were conducted to observe the behavior of the grouping hyper-heuristic framework considering different k values for each problem instance, as specified in tables 6.1, 6.2 and 6.3, while the heuristic selection is fixed from {SR, RL, DH} and the heuristic acceptance is fixed from {ILTA, LACC, GDEL}. A thorough performance analysis of the hyper-heuristics is performed. Then the performance of the hyper-heuristic with the best mean corner point is compared to the performance of some previously proposed approaches.

6.4 Results and Discussion for Graph coloring

For each one of the graph coloring problem instances given in Table 6.1, the success rate of each hyper-heuristic approach is calculated for all the k values in the given range in the table. For example, the best known coloring for the DSJC500.1 problem instance is 12; and all the hyper-heuristic approaches are tested with $7 \leq k \leq 17$, and hence the success rate of finding a coloring with each value of $k \in [7, 17]$ is calculated. Tables 6.4, 6.5 and 6.6 show the success rate of each hyper-heuristic approach with each problem instance for different values of k selected from the given range. For instance, in Table 6.4, out of the 30 runs performed using the RL-ILTA hyper-heuristic on DSJC500.1 problem instance, the percentage of the runs in which solutions with the best colorings

TABLE 6.4: The Performance of Reinforcement Learning (RL) Selection Hyper-heuristics: the success rate ($sRate\%$) and the average best coloring ($\mu(k_{best})$) of each hyper-heuristic approach on the graph coloring problem instances over the 30 runs.

			RL-ILTA		RL-LACC		RL-GDEL	
	Instance	k	sRate%	$\mu_t(s)$	sRate%	$\mu_t(s)$	sRate%	$\mu_t(s)$
COLOR02	myciel3	4	100.00	0.003	100.00	0.004	93.33	0.002
	myciel4	5	100.00	0.005	100.00	0.005	100.00	0.006
	myciel5	6	100.00	0.009	100.00	0.011	100.00	0.069
	queen5.5	5	100.00	0.034	90.00	0.037	100.00	0.194
	queen6.6	7	100.00	0.786	20.00	1.449	100.00	111.831
	queen7.7	7	100.00	0.900	16.67	0.040	100.00	39.350
	queen8.8	9	100.00	4.110	3.33	0.150	100.00	74.530
DIMACS	DSJC125.1	5	96.67	113.20	6.67	6.71	100.00	295.58
		6	100.00	0.09	100.00	0.17	100.00	55.02
		7	100.00	0.03	100.00	0.04	100.00	50.18
	DSJC125.5	17	33.33	470.96	0.00	–	0.00	–
		18	100.00	5.68	60.00	115.70	66.67	655.66
		19	100.00	0.85	100.00	1.80	100.00	82.63
	DSJC125.9	44	100.00	40.02	80.00	69.40	0.00	–
		45	100.00	4.50	100.00	27.05	10.00	538.47
		46	100.00	1.58	100.00	3.90	86.67	422.53
	DSJC250.1	8	10.00	1131.10	0.00	–	0.00	–
		9	100.00	2.35	100.00	7.46	100.00	194.28
		10	100.00	0.44	100.00	0.60	100.00	175.16
	DSJC250.5	28	0.00	–	0.00	–	0.00	–
		29	20.00	2132.40	0.00	–	0.00	–
		30	100.00	288.20	20.00	1164.80	0.00	–
	DSJC250.9	72	10.00	3301.23	0.00	–	0.00	–
		73	100.00	811.32	40.00	886.41	0.00	–
		74	100.00	177.44	80.00	648.03	0.00	–
	DSJC500.1	12	0.00	–	0.00	–	0.00	–
		13	60.00	1115.87	20.00	1331.99	0.00	–
		14	100.00	26.49	100.00	43.70	0.00	–
	DSJC500.5	48	0.00	–	0.00	–	0.00	–
		49	0.00	–	0.00	–	0.00	–
		50	0.00	–	0.00	–	0.00	–
		52	10.00	1797.03	0.00	–	0.00	–
	le450_25a	25	100.00	7.85	100.00	25.33	0.00	–
		26	100.00	1.56	100.00	2.81	83.33	472.34
		27	100.00	0.82	100.00	1.94	100.00	109.32
	le450_25b	25	100.00	11.85	100.00	6.22	6.67	569.68
		26	100.00	1.19	100.00	2.55	100.00	223.61
		27	100.00	0.65	100.00	1.66	100.00	73.37
	le450_25c	25	0.00	–	0.00	–	0.00	–
		26	0.00	–	0.00	–	0.00	–
		27	76.67	565.69	40.00	873.93	0.00	–
	le450_25d	25	0.00	–	0.00	–	0.00	–
		26	0.00	–	0.00	–	0.00	–
		27	70.00	654.47	30.00	906.22	0.00	–

TABLE 6.5: The Performance of Adaptive Dynamic Heuristics Set (ADHS) Selection Hyper-heuristics: the success rate ($sRate\%$) and the average best coloring ($\mu(k_{best})$) of each hyper-heuristic approach on the graph coloring problem instances over the 30 runs.

			ADHS-ILTA		ADHS-LACC		ADHS-GDEL	
	Instance	k	sRate%	$\mu_t(s)$	sRate%	$\mu_t(s)$	sRate%	$\mu_t(s)$
COLOR02	myciel3	4	100.00	0.006	100.00	0.006	96.67	0.004
	myciel4	5	100.00	0.015	100.00	0.018	100.00	0.016
	myciel5	6	100.00	0.044	100.00	0.054	100.00	3.592
	queen5.5	5	100.00	0.167	100.00	1.053	100.00	5.532
	queen6.6	7	56.67	290.060	43.33	86.180	93.33	204.640
	queen7.7	7	50.00	304.040	10.00	432.160	86.67	152.420
	queen8.8	9	60.00	305.880	26.67	224.680	86.67	335.830
DIMACS	DSJC125.1	5	30.00	618.95	6.67	112.04	90.00	533.55
		6	100.00	0.48	100.00	12.93	100.00	39.25
		7	100.00	0.16	100.00	0.37	100.00	32.14
	DSJC125.5	17	13.33	821.69	0.00	–	0.00	–
		18	93.33	228.10	76.67	146.32	60.00	562.04
		19	100.00	14.66	100.00	12.02	96.67	226.22
	DSJC125.9	44	86.67	146.55	63.33	183.39	0.00	–
		45	100.00	29.08	93.33	32.98	10.00	537.80
		46	100.00	9.76	100.00	7.67	43.33	464.38
	DSJC250.1	8	30.00	1047.59	0.00	–	0.00	–
		9	100.00	71.23	100.00	59.16	100.00	355.09
		10	100.00	1.99	100.00	1.90	100.00	168.32
	DSJC250.5	28	0.00	–	0.00	–	0.00	–
		29	20.00	3042.47	0.00	–	0.00	–
		30	80.00	1704.74	40.00	913.88	0.00	–
	DSJC250.9	72	0.00	–	0.00	–	0.00	–
		73	60.00	2281.95	50.00	1357.03	0.00	–
		74	90.00	865.49	90.00	667.89	0.00	–
	DSJC500.1	12	0.00	–	0.00	–	0.00	–
		13	70.00	1115.84	80.00	496.77	0.00	–
		14	100.00	23.69	100.00	53.11	0.00	–
	DSJC500.5	48	0.00	–	0.00	–	0.00	–
		49	0.00	–	0.00	–	0.00	–
		50	0.00	–	0.00	–	0.00	–
		52	3.33	2553.04	0.00	–	0.00	–
	le450_25a	25	100.00	85.82	96.67	36.19	3.33	1019.41
		26	100.00	3.15	100.00	5.27	86.67	345.32
		27	100.00	1.80	100.00	3.38	100.00	147.13
	le450_25b	25	100.00	7.58	100.00	29.14	20.00	649.71
		26	100.00	2.13	100.00	3.75	100.00	211.16
		27	100.00	1.40	100.00	3.16	100.00	101.29
	le450_25c	25	0.00	–	0.00	–	0.00	–
		26	0.00	–	0.00	–	0.00	–
		27	83.33	626.85	30.0	2346.70	0.00	–
	le450_25d	25	0.00	–	0.00	–	0.00	–
		26	0.00	–	0.00	–	0.00	–
		27	63.33	704.38	36.67	665.24	0.00	–

TABLE 6.6: The Performance of Simple Random (SR) Selection Hyper-heuristics: the success rate ($sRate\%$) and the average best coloring ($\mu(k_{best})$) of each hyper-heuristic approach on the graph coloring problem instances over the 30 runs.

			SR-ILTA		SR-LACC		SR-GDEL	
	Instance	k	sRate%	$\mu_t(s)$	sRate%	$\mu_t(s)$	sRate%	$\mu_t(s)$
COLOR02	myciel3	4	100.00	0.007	100.00	0.004	100.00	0.004
	myciel4	5	100.00	0.007	100.00	0.003	100.00	0.008
	myciel5	6	100.00	0.011	100.00	0.015	100.00	0.398
	queen5.5	5	100.00	0.016	96.67	0.017	100.00	0.687
	queen6.6	7	33.33	0.157	20.00	0.780	100.00	83.140
	queen7.7	7	16.67	0.261	3.33	0.038	100.00	43.669
	queen8.8	9	30.00	2.032	10.00	1.232	100.00	69.154
DIMACS	DSJC125.1	5	3.33	450.25	0.00	–	96.67	374.43
		6	100.00	0.13	100.00	0.14	100.00	53.92
		7	100.00	0.06	100.00	0.07	100.00	51.31
	DSJC125.5	17	20.00	756.87	0.00	–	0.00	–
		18	93.33	117.65	53.33	41.71	90.00	480.85
		19	100.00	1.06	100.00	1.88	100.00	92.52
	DSJC125.9	44	90.00	72.28	86.67	57.01	0.00	–
		45	100.00	13.79	100.00	5.28	23.33	340.94
		46	100.00	1.87	100.00	3.20	96.67	432.36
	DSJC250.1	8	0.00	–	0.00	–	0.00	–
		9	100.00	5.58	100.00	4.36	100.00	176.98
		10	100.00	0.57	100.00	0.74	100.00	175.51
	DSJC250.5	28	0.00	–	0.00	–	0.00	–
		29	0.00	–	0.00	–	0.00	–
		30	60.00	1350.55	20.00	650.08	0.00	–
	DSJC250.9	72	0.00	–	0.00	–	0.00	–
		73	50.00	2264.24	60.00	1547.57	0.00	–
		74	100.00	649.87	90.00	668.56	0.00	–
	DSJC500.1	12	0.00	–	0.00	–	0.00	–
		13	30.00	1005.16	40.00	913.15	0.00	–
		14	100.00	27.99	100.00	30.55	0.00	–
	DSJC500.5	48	0.00	–	0.00	–	0.00	–
		49	0.00	–	0.00	–	0.00	–
		50	0.00	–	0.00	–	0.00	–
		53	3.33	1944.04	0.00	–	0.00	–
	le450_25a	25	100.00	6.36	100.00	33.61	0.00	–
		26	100.00	1.52	100.00	3.34	76.67	518.65
		27	100.00	0.89	100.00	2.52	100.00	99.39
	le450_25b	25	100.00	3.10	100.00	7.70	0.00	–
		26	100.00	1.16	100.00	3.03	100.00	221.37
		27	100.00	0.67	100.00	1.95	100.00	83.17
	le450_25c	25	0.00	–	0.00	–	0.00	–
		26	0.00	–	0.00	–	0.00	–
		27	40.00	1395.13	16.67	783.49	0.00	–
	le450_25d	25	0.00	–	0.00	–	0.00	–
		26	0.00	–	0.00	–	0.00	–
		27	60.00	2326.19	30.00	2230.95	0.00	–

($k = 12$) were found is 0%; while the percentage of the runs in which solutions with $k = 13$ colors are found is 60% and the percentage of the runs in which solutions with $k = 14$ colors are found is 100%. This means that for this particular problem instance, the RL-ILTA hyper-heuristic failed to find a zero conflict solution with 12 colors in all the 30 runs that were performed, but it succeeded in finding a zero conflict solution with 13 colors in 60% of the runs, and a zero conflict solution with 14 colors in each one of the 30 runs. These tables also show the average time (in seconds) taken to achieve those success rates. Only the durations of the successful runs were taken into consideration when these average times were calculated.

From these three tables, it can be seen that most of the tested hyper-heuristic approaches managed to successfully find the best colorings of the selected COLOR02 problem instances in a short period of time. Hence, for these instances, only the success rates and the average times for the best known value of k are provided. The RL-ILTA and the SR-GDEL hyper-heuristics successfully found the best coloring for each problem instance in each one of the 30 runs, achieving 100.0% success rates across the board, with a maximum average time of 4.1 seconds for the RL-ILTA approach and 83.14 seconds for the SR-GDEL. On the other hand, all hyper-heuristics that use the LACC acceptance method managed to find the best colorings for some problem instances in each one of the 30 runs, but failed to find the best colorings for the rest of the problem instances in most of the runs. For example, the SR-LACC and the RL-LACC hyper-heuristics both have a success rate of 100% with each one of the myciel3, myciel4 and myciel5 instances, but also a success rate of only 3.33% with queen7.7 and queen8.8 problem instances, respectively. Apart from the LACC-based hyper-heuristics, the only other hyper-heuristic that has a success rate below 50% with any of the COLOR02 problem instances is the SR-ILTA, which achieved success rates of 33.33, 16.70 and 30.00 with queesn6.6, queen7.7 and queen8.8, respectively.

The performances of the hyper-heuristic approaches were much varied and less successful when applied on the selected DIMACS problem instances. Also, much longer periods of time were needed to find the best solutions in most of the cases. Generally, hyper-heuristics that use the ILTA acceptance have achieved better success rates than hyper-heuristics that use LACC and GDEL acceptance methods in most of the cases. ILTA-based hyper-heuristics have outperformed their LACC and GDEL counterparts in all problem instances for all values of k with the exception of DSJC125.1 at $k = 5$, DSJC250.9 at $k = 73$ and DSJC500.1 at $k = 13$. On the other hand, hyper-heuristics that use GDEL acceptance method have the worst average times and success rates on most of the DIMACS instances compared to the rest of the hyper-heuristics. Also, for problem instances with the same number of nodes, there is a positive correlation between the number of edges in the instance on one hand and the success rate and the

average time on the other hand, respectively. For instance, each one of the four a, b, c, and d le450_25 instances has the same number of nodes of 450 nodes, but has a different number of edges of 8260, 8263, 17343 and 17425 edges, respectively. As a result, for almost all the hyper-heuristic approaches, the success rates and average times achieved in le450_25a and le450_25b are much better than the success rates and average times achieved in le450_25c and le450_25d.

TABLE 6.7: The average best colorings and the standard deviations of reinforcement learning based hyper-heuristics on graph coloring instances across the 30 runs.

		RL-ILTA		RL-LACC		RL-GDEL	
Instance	k^*	$\mu(k_{best})$	$\sigma(k_{best})$	$\mu(k_{best})$	$\sigma(k_{best})$	$\mu(k_{best})$	$\sigma(k_{best})$
myciel3	4	**4.0**	±0.0	**4.0**	±0.0	4.07	±0.25
myciel4	5	**5.0**	±0.0	**5.0**	±0.0	**5.0**	±0.0
myciel5	6	**6.0**	±0.0	**6.0**	±0.0	**6.0**	±0.0
queen5.5	5	**5.0**	±0.0	5.1	±0.31	**5.0**	±0.0
queen6.6	7	**7.0**	±0.0	7.8	±0.41	**7.0**	±0.0
queen7.7	7	**7.0**	±0.0	8.4	±0.77	**7.0**	±0.0
queen8.8	9	**9.0**	±0.0	9.97	±0.18	**9.0**	±0.0
DSJC125.1	5	5.03	±0.18	5.93	±0.25	**5.0**	±0.0
DSJC125.5	17	**17.67**	±0.48	18.4	±0.50	18.33	±0.48
DSJC125.9	44	**44.0**	±0.0	44.2	±0.41	46.03	±0.49
DSJC250.1	8	**8.9**	±0.30	9.0	±0.0	9.0	±0.0
DSJC250.5	28	**29.8**	±0.41	30.8	±0.41	32.6	±0.50
DSJC250.9	72	**72.9**	±0.31	73.8	±0.76	79.4	±0.50
DSJC500.1	12	**13.4**	±0.50	13.8	±0.41	15.5	±0.51
DSJC500.5	48	**53.6**	±0.67	56.5	±0.51	57.4	±0.50
le450_25a	25	**25.0**	±0.0	**25.0**	±0.0	26.17	±0.38
le450_25b	25	**25.0**	±0.0	**25.0**	±0.0	25.93	±0.25
le450_25c	25	**27.23**	±0.43	27.6	±0.50	32.6	±0.50
le450_25d	25	**27.3**	±0.47	27.7	±0.47	33.4	±0.50
Wins		**18**		5		7	

Tables 6.7, 6.8 and 6.9 show the average best coloring and standard deviation for each hyper-heuristic approach on each problem instance across the 30 runs. ±0.0 standard deviation corresponds to 100.0% success rate. The row denoted as 'Wins' in each table shows the number of problem instances in which the corresponding hyper-heuristic approach achieved the best average coloring including ties with the other algorithms. From these tables it can be seen that hyper-heuristics that use the ILTA acceptance method have the most number of wins compared to hyper-heuristics that use either the LACC or the GDEL acceptance methods. Hence the RL-ILTA hyper-heuristic has the most number of wins across all the tested approaches, this hyper-heuristic is used as a reference to carry out statistical tests in order to determine how significant the differences in the best colorings found by each one of the tested hyper-heuristic approaches with regard to the RL-ILTA approach are.

TABLE 6.8: The average best colorings and the standard deviations of adaptive dynamic heuristics set based hyper-heuristics on graph coloring instances across the 30 runs.

		ADHS-ILTA		ADHS-LACC		ADHS-GDEL	
Instance	k^*	$\mu(k_{best})$	$\sigma(k_{best})$	$\mu(k_{best})$	$\sigma(k_{best})$	$\mu(k_{best})$	$\sigma(k_{best})$
myciel3	4	**4.0**	±0.0	**4.0**	±0.0	4.1	±0.55
myciel4	5	**5.0**	±0.0	**5.0**	±0.0	**5.0**	±0.0
myciel5	6	**6.0**	±0.0	**6.0**	±0.0	**6.0**	±0.0
queen5.5	5	**5.0**	±0.0	**5.0**	±0.0	**5.0**	±0.0
queen6.6	7	7.43	±0.50	7.57	±0.50	**7.07**	±0.25
queen7.7	7	7.7	±0.79	8.37	±0.67	**7.17**	±0.46
queen8.8	9	9.4	±0.50	9.73	±0.45	**9.13**	±0.35
DSJC125.1	5	5.7	±0.47	5.93	±0.25	**5.1**	±0.31
DSJC125.5	17	**17.93**	±0.45	18.23	±0.43	18.43	±0.57
DSJC125.9	44	**44.13**	±0.35	44.43	±0.63	46.6	±0.86
DSJC250.1	8	**8.7**	±0.47	9.0	±0.0	9.0	±0.0
DSJC250.5	28	**30.0**	±0.64	30.6	±0.50	32.2	±0.89
DSJC250.9	72	**73.5**	±0.68	73.6	±0.67	80.57	±0.57
DSJC500.1	12	13.3	±0.47	**13.2**	±0.41	16.3	±0.65
DSJC500.5	48	**53.67**	±0.71	54.5	±0.82	58.77	±1.01
le450_25a	25	**25.0**	±0.0	25.03	±0.18	26.1	±0.40
le450_25b	25	**25.0**	±0.0	**25.0**	±0.0	25.8	±0.41
le450_25c	25	**27.17**	±0.38	27.7	±0.47	33.97	±0.81
le450_25d	25	**27.4**	±0.56	27.63	±0.49	34.13	±0.68
Wins		**14**		6		7	

Table 6.10 shows the pairwise performance comparison of the hyper-heuristics based on the Wilcoxon Signed Rank statistical test using the best colorings attained by RL-ILTA hyper-heuristic as the comparison reference. Almost all of the results obtained by the GDEL based hyper-heuristics are significantly worse than the results of the RL-ILTA hyper-heuristic, with the exception of RL-GDEL on DSJC125.1. In this particular case, the RL-GDEL hyper-heuristic performs significantly better than RL-ILTA. Similarly, the RL-ILTA hyper-heuristic performs either significantly or slightly better than the all the LACC based hyper-heuristics in almost all of the cases with the exception of DH-LACC on DSJC500.1. In this particular case, the DH-LACC hyper-heuristic performs significantly better than the RL-ILTA.

One of the observations is that the DH-ILTA and RL-ILTA hyper-heuristics deliver a competitive performance. On DSJC250.1, DSJC500.1 and le450_25c, DH-ILTA hyper-heuristic performs significantly better than the RL-ILTA hyper-heuristic. This is expected, because the DH-ILTA is known to be a very powerful algorithm in cross domain search Misir et al. [2013]. However, on more instances, namely DSJC125.1, DSJC125.9, DSJC250.9, DSJC500.5 and le450_25d, RL-ILTA outperforms DH-ILTA. This performance difference is statistically significant. Additionally, on 4 other instances, RL-ILTA performs slightly better than DH-ILTA. They have a tie on six benchmark instances.

TABLE 6.9: The average best colorings and the standard deviations of simple random based hyper-heuristics on graph coloring instances across the 30 runs.

		SR-ILTA		SR-LACC		SR-GDEL	
Instance	k^*	$\mu(k_{best})$	$\sigma(k_{best})$	$\mu(k_{best})$	$\sigma(k_{best})$	$\mu(k_{best})$	$\sigma(k_{best})$
myciel3	4	**4.0**	±0.0	**4.0**	±0.0	**4.0**	±0.0
myciel4	5	**5.0**	±0.0	**5.0**	±0.0	**5.0**	±0.0
myciel5	6	**6.0**	±0.0	**6.0**	±0.0	**6.0**	±0.0
queen5.5	5	**5.0**	±0.0	5.03	±0.18	**5.0**	±0.0
queen6.6	7	7.67	±0.48	7.8	±0.41	**7.0**	±0.0
queen7.7	7	8.23	±0.73	8.53	±0.57	**7.0**	±0.0
queen8.8	9	9.7	±0.47	9.9	±0.31	**9.0**	±0.0
DSJC125.1	5	5.97	±0.18	6.0	±0.0	**5.03**	±0.18
DSJC125.5	17	**17.87**	±0.51	18.47	±0.51	18.1	±0.31
DSJC125.9	44	**44.1**	±0.31	44.13	±0.35	45.8	±0.48
DSJC250.1	8	**9.0**	±0.0	**9.0**	±0.0	**9.0**	±0.0
DSJC250.5	28	**30.4**	±0.50	30.9	±0.55	31.9	±0.55
DSJC250.9	72	**73.5**	±0.51	**73.5**	±0.68	80.5	±0.51
DSJC500.1	12	13.7	±0.47	**13.6**	±0.50	16.9	±0.71
DSJC500.5	48	**54.57**	±0.73	54.9	±0.71	59.0	±0.64
le450_25a	25	**25.0**	±0.0	**25.0**	±0.0	26.23	±0.43
le450_25b	25	**25.0**	±0.0	**25.0**	±0.0	26.0	±0.0
le450_25c	25	**27.6**	±0.50	27.83	±0.38	34.13	±0.73
le450_25d	25	**27.4**	±0.50	27.7	±0.47	34.2	±0.61
Wins		14		8		9	

To better evaluate the differences between the performances of the 9 hyper-heuristics on the selected problem instances, another performance comparison was carried out using the hyper-volume measure, in which the performance of different algorithms is compared in terms of the size of the search space that is covered by the final Pareto front of each hyper-heuristic. The calculation of the hyper-volume measure in this study was simplified using the fact that the difference in the number of groups between any consecutive two points in the final non-dominated set is always 1, as explained in section 6.2. Figure 6.3 shows the box plots of the 30 hyper-volume values produced by each hyper-heuristic on each problem instance. The figure clearly exposes the fact that hyper-heuristics that use the GDEL acceptance method, denoted as 3 in the figure, cover the least size of the search space compared to the other hyper-heuristics in most of the selected data sets. This observation is confirmed by Figure 6.4 which shows the box plots of the best number of colors achieved by each hyper-heuristic approach on selected DIMACS data sets. From this figure it can be seen that the hyper-heuristics that use the GDEL acceptance method, denoted as 3 in the figure, have found a worse number of colors in most of the cases compared the other hyper-heuristic approaches.

TABLE 6.10: Wilcoxon Signed Rank statistical test for the graph coloring results using the RL-ILTA hyper-heuristic as a reference for the comparison: '>' ('<') denotes that RL-ILTA is significantly better (worse) than the corresponding approach in that column, '≥' means that RL-ILTA is slightly better, and '=' means that there is no difference between the two hyper-heuristic approaches across the 30 runs.

Instance	RL LACC	RL GDEL	ADHS ILTA	ADHS LACC	ADHS GDEL	SR ILTA	SR LACC	SR GDEL
myciel3	=	>	=	=	>	=	=	=
myciel4	=	=	=	=	=	=	=	=
myciel5	=	=	=	=	=	=	=	=
queen5.5	>	=	=	=	=	=	>	=
queen6.6	>	=	≥	>	>	>	>	=
queen7.7	>	=	>	>	>	>	>	=
queen8.8	>	=	≥	>	>	>	>	=
DSJC125.1	>	<	>	>	>	>	>	=
DSJC125.5	>	>	≥	>	>	≥	>	≥
DSJC125.9	≥	>	>	≥	>	>	>	>
DSJC250.1	>	>	<	>	>	>	>	>
DSJC250.5	>	>	≥	>	>	>	>	>
DSJC250.9	>	>	>	>	>	>	>	>
DSJC500.1	≥	>	<	<	>	≥	≥	>
DSJC500.5	>	>	>	>	>	>	>	>
le450_25a	=	>	=	>	>	=	=	>
le450_25b	=	>	=	=	>	=	=	>
le450_25c	≥	>	<	>	>	≥	>	>
le450_25d	≥	>	>	≥	>	>	≥	>

6.4.1 Performance Comparison to Previously Proposed Graph coloring Algorithms

In Table 6.11, the performance of the hyper-heuristic approach that was found to be the best, namely the RL-ILTA hyper-heuristic, is compared to some previously proposed approaches from the literature for graph coloring. In this table, the results denoted as RL-ILTA are the best colorings obtained in the experiments performed in this study. The results under M-LLE are obtained by a multi-objective genetic grouping algorithm described in Korkmaz [2010]. Lowest Index Max Crossover (LIMX), Greedy Partition Crossover Lowest Index (GPX-LI) and Greedy Partition Crossover Cardinality Based (GPX-CB) graph coloring algorithms are proposed in Ülker et al. [2006]. Yılmaz and Korkmaz [2010] proposed two modified versions of the LLE representation which are Linear Linkage Encoding With Ending Node Links (LLE-e) and Linear Linkage Encoding With Backward Links (LLE-b), and both of them are tested using genetic operators. This last study also tested these operators with classical Linear Linkage Encoding (LLE). The results int the last two columns denoted (Kir-B) and (Kir-C) are graph coloring algorithms proposed in Kirovski and Potkonjak [1998]. Fields marked as '−' means that

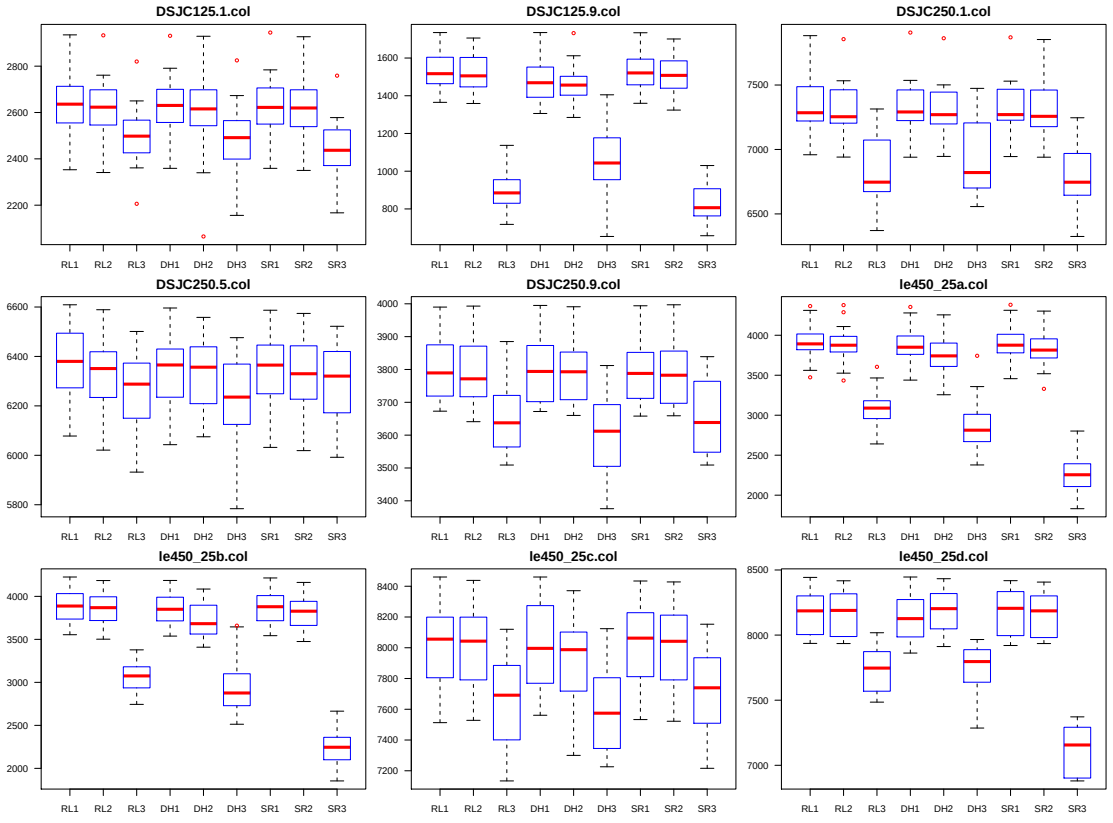

FIGURE 6.3: Box plots of the hyper-volume value of all the final Pareto fronts achieved by each hyper-heuristic approach on selected DIMACS data sets. 1, 2 and 3 in the hyper-heuristic approaches names refer to ILTA, LACC and GDEL acceptance methods respectively

the solution for that problem instance with that specific algorithm is not reported. The 'wins' row shows the number of instances in which the corresponding approach hit the best known coloring. As it can clearly be seen in the table, the RL-ILTA approach is not only competitive with the previous algorithms, but it also outperforms the previously proposed approaches almost in all of the cases.

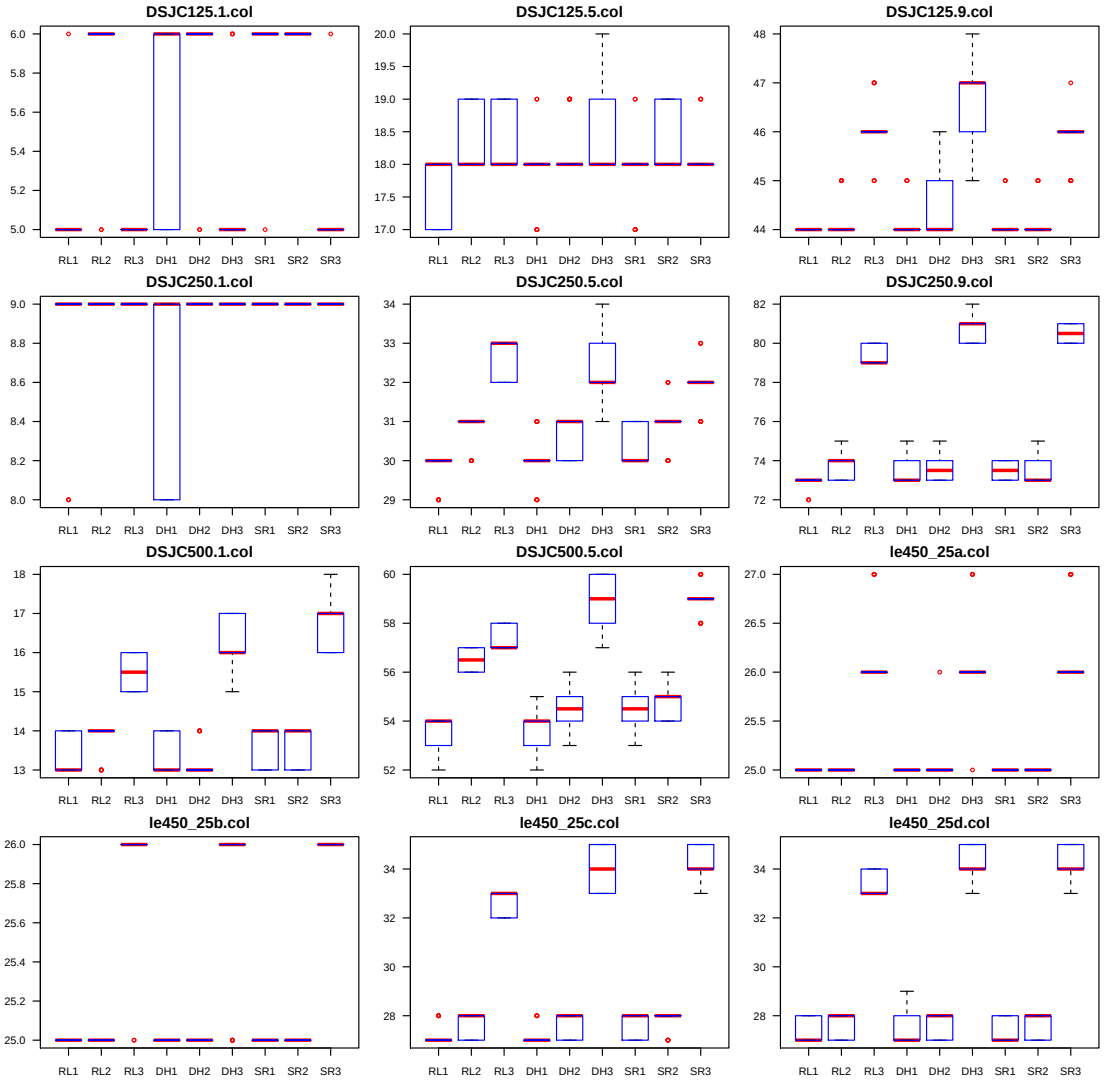

FIGURE 6.4: Box plots of the best number of colors achieved by each hyper-heuristic approach on selected DIMACS data sets. 1, 2 and 3 in the hyper-heuristic approaches names refer to ILTA, LACC and GDEL acceptance methods respectively

TABLE 6.11: Comparing the performances of different approaches on graph coloring problem instances based on the best number of colors. The entries in bold indicate the best result obtained by the associated algorithm for the given instance.

Instance	k^*	RL-ILTA	M-LLE	LIMX	GPX-LI	GPX-CB	LLE-e	LLE-b	LLE	Kir-B	Kir-C
DSJC125.1	5	**5**	–	–	–	–	–	–	–	–	–
DSJC125.5	17	**17**	18	18	18	18	19	19	18	19	18
DSJC125.9	44	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	45	45
DSJC250.1	8	**8**	9	9	9	9	9	9	9	9	9
DSJC250.5	28	**29**	–	31	31	31	–	–	–	30	30
DSJC250.9	72	**72**	75	75	75	74	74	74	74	77	77
le450_25a	25	**25**	–	**25**	**25**	**25**	–	–	–	**25**	**25**
le450_25b	25	**25**	–	**25**	**25**	**25**	–	–	–	**25**	**25**
le450_25c	25	**27**	29	28	28	28	29	29	29	28	28
le450_25d	25	**27**	–	28	28	28	–	–	–	–	–
DSJC500.1	12	**13**	–	14	14	14	–	–	–	14	14
DSJC500.5	48	**52**	55	–	–	–	–	–	–	–	–
Wins		**12**	1	3	3	3	1	1	1	2	2

6.5 Results and Discussion for Examination Timetabling

Similarly, the grouping hyper-heuristics were applied on the selected timetabling instances given in Table 6.2. The result are shown in Tables 6.12, 6.13 and 6.14. For each one of the timetabling problem instances, the success rate of each hyper-heuristic approach is calculated for all the k values in the given range in the table in the same fashion used for the graph coloring instances. The main difference is that, in Tables 6.12, 6.13 and 6.14, these values are shown only for the best number of groups in each instance. These tables also show the average best coloring and standard deviation for each hyper-heuristic approach on each problem instance across the 30 runs. A standard deviation of ± 0.0 corresponds to 100.0% success rate, indicating that the corresponding hyper-heuristic approach managed to find the best grouping for that particular instance in each of the 30 runs performed. The row denoted as 'wins' in each table shows the number of problem instances in which the corresponding hyper-heuristic approach achieved the best average coloring including ties with the other algorithms.

From these three tables, it can be seen that all hyper-heuristics that use the LACC or the GDEL acceptance methods failed to find the best grouping in many cases, such as the RL-LACC and the SR-GDEL on car91I, car92 and uta92I instances. In fact, all the LACC and GDEL-based hyper-heuristics have always achieved a success rate that is less than or equal to 40%, apart from when applied on three instances, namely hec92, sta83 and yor83. On the other hand, all the ILTA-based hyper-heuristics have managed to successfully find the best grouping for each one of the problem instances in at least one of the 30 runs performed, apart from the SR-ILTA when applied on the car91I instance.

TABLE 6.12: The Performance of Reinforcement Learning (RL) Selection Hyper-heuristics: the success rate ($sRate\%$), the average best coloring ($\mu(k_{best})$) and the standard deviation ($\sigma(k_{best})$) of each hyper-heuristic approach on the timetabling problem instances over the 30 runs.

			RL-ILTA			RL-LACC			RL-GDEL		
	Instance	k^*	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$
Toronto	hec92	17	**100.00**	**17.0**	±0.0	83.33	17.17	±0.38	86.67	17.13	±0.35
	sta83	13	**100.00**	**13.0**	±0.0	**100.00**	**13.0**	±0.0	**100.00**	**13.0**	±0.0
	yor83	19	**16.67**	**19.83**	±0.38	10.00	20.0	±0.45	6.67	20.27	±0.74
	ute92	10	**100.00**	**10.0**	±0.0	**100.00**	**10.0**	±0.0	**100.00**	**10.0**	±0.0
	ear83	24	**33.33**	**24.83**	±0.7	23.33	25.17	±1.02	13.33	25.6	±0.93
	tre92	23	6.67	**24.0**	±1.11	36.67	**24.0**	±0.98	**40.00**	24.07	±1.14
	lse91	18	**63.33**	**18.77**	±1.14	23.33	19.33	±1.12	33.33	19.23	±1.1
	kfu93	20	**33.33**	**21.37**	±1.3	13.33	21.93	±1.36	16.67	21.93	±1.34
	rye93	21	**53.33**	**21.47**	±0.51	20.00	22.03	±0.67	13.33	22.17	±0.65
	car92	32	**6.67**	**34.17**	±1.18	0.00	34.43	±1.14	0.00	34.6	±1.13
	uta92 I	35	**13.33**	**36.43**	±0.9	0.00	36.97	±1.07	3.33	36.9	±1.12
	car91 I	35	**3.33**	**36.5**	±0.73	0.00	37.33	±1.09	0.00	37.33	±1.09
	Wins		**11**	**12**	–	2	3	–	3	2	–

TABLE 6.13: The Performance of Adaptive Dynamic Heuristics Set (ADHS) Selection Hyper-heuristics: the success rate ($sRate\%$), the average best coloring ($\mu(k_{best})$) and the standard deviation ($\sigma(k_{best})$) of each hyper-heuristic approach on the timetabling problem instances over the 30 runs.

			ADHS-ILTA			ADHS-LACC			ADHS-GDEL		
	Instance	k^*	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$
Toronto	hec92	17	**100.00**	**17.0**	±0.0	90.00	17.1	±0.31	93.33	17.07	±0.25
	sta83	13	**100.00**	**13.0**	±0.0	96.67	13.03	±0.18	93.33	13.07	±0.25
	yor83	19	**43.33**	**19.63**	±0.61	30.00	19.9	±0.71	6.67	20.13	±0.51
	ute92	10	**100.00**	**10.0**	±0.0	**100.00**	**10.0**	±0.0	96.67	10.03	±0.18
	ear83	24	**43.33**	**24.73**	±0.74	23.33	25.1	±0.84	3.33	25.3	±0.65
	tre92	23	**56.67**	**23.77**	±1.01	33.33	24.17	±1.12	20.00	24.4	±1.13
	lse91	18	**56.67**	**18.87**	±1.14	40.00	19.03	±1.07	26.67	19.17	±0.99
	kfu93	20	**53.33**	**21.23**	±1.5	6.67	21.9	±1.21	23.33	21.93	±1.44
	rye93	21	**56.67**	**21.4**3	±0.5	13.33	22.23	±0.86	6.67	22.3	±0.75
	car92	32	**13.33**	**34.13**	±1.25	3.33	34.2	±1.03	6.67	34.4	±1.22
	uta92 I	35	**10.00**	**36.4**	±0.86	3.33	36.7	±0.84	0.00	37.03	±0.96
	car91 I	35	**6.67**	**36.57**	±0.94	0.00	37.63	±1.22	0.00	36.83	±0.99
	Wins		**12**	**12**	–	0	1	–	0	0	–

TABLE 6.14: The Performance of Simple Random (SR) Selection Hyper-heuristics: the success rate ($sRate\%$), the average best coloring ($\mu(k_{best})$) and the standard deviation ($\sigma(k_{best})$) of each hyper-heuristic approach on the timetabling problem instances over the 30 runs.

			SR-ILTA			SR-LACC			SR-GDEL		
	Instance	k^*	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$
Toronto	hec92	17	**96.67**	**17.03**	±0.18	86.67	17.13	±0.35	73.33	17.27	±0.45
	sta83	13	**100.00**	**13.0**	±0.0	96.67	13.03	±0.18	86.67	13.13	±0.35
	yor83	19	3.33	**20.57**	±0.86	**6.67**	20.83	±1.15	0.00	21.37	±1.22
	ute92	10	**100.00**	**10.0**	±0.0	93.33	10.07	±0.25	96.67	10.03	±0.18
	ear83	24	**40.00**	**24.8**	±0.85	6.67	25.5	±0.82	6.67	25.37	±0.61
	tre92	23	**33.33**	**24.3**	±1.32	6.67	24.73	±1.14	3.33	24.9	±1.18
	lse91	18	**43.33**	**18.97**	±1.13	26.67	19.33	±1.27	16.67	19.5	±1.14
	kfu93	20	**26.67**	**21.83**	±1.44	10.00	22.0	±1.14	16.67	22.0	±1.44
	rye93	21	**23.33**	**21.77**	±0.43	0.00	23.03	±0.89	6.67	22.57	±0.9
	car92	32	**13.33**	**34.33**	±1.32	0.00	34.43	±1.14	0.00	34.7	±1.02
	uta92 I	35	**6.67**	**36.77**	±1.07	0.00	37.07	±1.01	0.00	37.17	±1.12
	car91 I	35	0.00	**36.4**	±0.56	0.00	37.3	±1.09	0.00	36.97	±1.03
	Wins		**10**	**12**	–	1	0	–	0	0	–

6.5.1 Performance Comparison to Previously Proposed Examination Timetabling Algorithms

Also, from these tables it can be seen that hyper-heuristics that use the ILTA acceptance method have the most number of wins in terms of the success rate as well as the average best grouping compared to hyper-heuristics that use either the LACC or the GDEL acceptance methods. The ADHS-ILTA hyper-heuristic has the most number of wins across all the tested approaches. Hence, the performance of ADHS-ILTA is compared against the best results obtained by some previously proposed timetabling approaches including Carter et al. [1996] denoted as (Carter), Caramia et al. [2001] denoted as (Caramia) and Merlot et al. [2002] denoted as (Merlot). The approaches in Ülker et al. [2006] including Greedy Partition Crossover Lowest Index, denoted as (GPX-LI), Greedy Partition Crossover Cardinality Based, denoted as (GPX-CB) and Lowest Index Max Crossover, denoted as (LIMX), are also considered in our comparison. This comparison is shown in Table 6.15.

Carter et al. [1996] employed known graphcoloring techniques to tackle examination-timetabling without side constraints. They solved an underlying graph coloringproblem, where examinations were sorted in order of decreasing difficulty according to some criterion: examinations were assigned to time slots without creatingconflicts, in the order they were processed. To reduce the length of the schedule obtained, some form of backtracking was used. Caramia et al. [2001] method starts by employing a greedy scheduler which considers exams by their non-increasing priorities and eventually updates the penalties based on the schedule found. Then a penalty decreaser attempts to improve the quality of the solution without increasing the number of the total number of time slots used in the current schedule. When the penalty decreaser is not successful, a penalty trader that tries to trade off penalties for time slots is invoked. Merlot et al. [2002] is a hybrid algorithm that consists of three phases. In the first phase, an initial solution is created using constraint programming. In the second phase, the initial solution is improved using simulated annealing; and in the third phase, a hill climber is employed for further improvement. The idea of the GPX is to repeatedly transmit the largest group from one parent, and to delete the vertices in this largest group from the other parent until all of the vertices are assigned to the child. The GPX-LI and the GPX-CB are two forms of the GPX that follow the rules of Cardinality and Lowest Index Ordering. In the GPX-LI, the groups with lower index numbers are given lower ids, whereas in the GPX-CB, the lower ids are assigned to the groups with higher cardinality. On the other hand, in LIMX large groups are transmitted to preserve Cardinality Based Ordering, and also groups beginning with lowest index number are transmitted

to preserve Lowest Index Ordering. Therefore this method can be considered as an amalgamate of teh two methods above.

Table 6.15 shows that the performance of ADHS-ILTA is competitive. It outperforms Merlot, GPX-LI, GPX-CB and LIMX algorithms, and performs as good as the Carter and Caramia approaches in almost half of the tested instances. ADHS-ILTA wins on 5 instances by providing a matching performance to previous best know results for hec92, sta83, yor83, ute92 and rye93 instances.

TABLE 6.15: Comparing the performances of different approaches on timetabling problem instances based on the best number of groups. The entries in bold indicate the best result obtained by the associated algorithm for the given instance.

Instance	ADHS-ILTA	LIMX	GPX-LI	GPX-CB	Carter	Caramia	Merlot
hec92	**17**	**17**	**17**	**17**	**17**	**17**	18
sta83	**13**	**13**	14	14	**13**	**13**	**13**
yor83	**19**	20	20	20	**19**	**19**	23
ute92	**10**	**10**	**10**	**10**	**10**	**10**	11
ear83	24	23	24	23	**22**	**22**	24
tre92	23	21	21	21	**20**	**20**	21
lse91	18	**17**	18	18	**17**	**17**	18
kfu93	20	20	20	20	**19**	**19**	21
rye93	**21**	23	23	23	**21**	**21**	22
car92	32	36	36	36	**28**	**28**	31
uta92	35	38	38	38	32	**30**	32
car91	35	36	37	35	**28**	**28**	30
Wins	5	4	2	2	11	**12**	1

6.6 Results and Discussion for Data Clustering

A final set of experiments with the same settings is performed on a set of data clustering benchmark problem instances described previously in Table 6.3. The results are summarised in the Tables 6.16, 6.17 and 6.18, showing the average best coloring, the standard deviation and the success rate for each of the grouping hyper-heuristics for each of the instances over 30 runs. The 'wins' in the bottom row of the tables shows the number of times each grouping selection hyper-heuristic has achieved the best average coloring including the ties with the other algorithms. A standard deviation of ± 0.0 for a particular algorithm indicates that this algorithm succeeded to find the best grouping for the particular instance over the 30 runs and achieved a 100% success rate. A similar outcome is observed for the performance of the grouping hyper-heuristics as in the problem domains reported in sections 6.4 and 6.5. Any selection method combined with ILTA performs better than the others, in general. Any grouping hyper-heuristic which uses LACC or GDEL as acceptance method achieved low success rates in most of the

instances, scoring a success rate that is less than 40% in most of the real and Gaussian instances. ADHS-ILTA receives the most number of wins across all the tested approaches, while RL-ILTA and SR-ILTA follows it in that order, respectively, as evidenced in the tables. ADHS-ILTA delivers the best success rate on all Gaussian instances, and most of the remaining instances. On average, ADHS-ILTA still performs better than the other grouping hyper-heuristics on 50% of instances and the standard deviation associated with the average values is the lowest in most of the cases. Hence, ADHS-ILTA is taken under consideration for further performance comparison to previously proposed approaches.

TABLE 6.16: The Performance of Reinforcement Learning Selection Hyper-heuristics: the success rate ($sRate\%$), the average best coloring ($\mu(k_{best})$) and the standard deviation ($\sigma(k_{best})$) of each hyper-heuristic approach on the data clustering problem instances over the 30 runs.

			RL-ILTA			RL-LACC			RL-GDEL		
	Instance	k^*	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$
Synthetic	Square1	4	**100.00**	**4.0**	±0.0	86.67	4.13	±0.35	93.33	4.07	±0.25
	Square4	4	**96.67**	**4.03**	±0.18	93.33	4.07	±0.25	86.67	4.13	±0.35
	Sizes5	4	**83.33**	3.9	±0.40	70.00	**4.03**	±0.56	73.33	**4.03**	±0.61
	Long1	4	**13.33**	**5.53**	±2.19	3.33	6.07	±2.07	3.33	5.83	±1.98
	Twenty	20	**23.33**	**20.87**	±1.36	16.67	20.9	±1.37	6.67	**20.87**	±1.38
	Fourty	40	**20.00**	**41.07**	±1.72	16.67	41.43	±1.73	10.00	41.33	±2.04
Gaussian	2D-4c	4	**16.67**	3.83	±0.59	6.67	**3.90**	±0.96	10.00	3.83	±0.83
	2D-10c	10	**13.33**	8.83	±1.21	3.33	8.97	±1.27	10.00	**9.17**	±1.46
	2D-20c	20	**63.33**	18.0	±2.30	40.00	18.23	±2.33	50.00	**18.73**	±2.35
	2D-40c	40	**20.00**	32.6	±3.24	10.00	32.4	±3.41	13.33	**32.77**	±3.21
	10D-4c	4	**13.33**	3.63	±1.07	6.67	3.77	±1.43	3.33	**4.13**	±1.72
	10D-10c	10	0.00	8.57	±1.74	0.00	8.5	±1.59	0.00	**8.9**	±1.86
Real	Zoo	7	**36.67**	**7.30**	±1.49	13.33	7.6	±1.61	16.67	7.67	±1.60
	Iris	3	**20.00**	**4.47**	±1.83	6.67	4.83	±1.84	3.33	4.90	±1.92
	Dermatology	6	**20.00**	**6.37**	±1.50	10.00	**6.37**	±1.52	3.33	6.57	±1.77
	Breast-cancer	2	**16.67**	**3.33**	±0.92	6.67	3.43	±0.90	6.67	3.43	±0.82
	Wins		**15**	**9**	–	0	3	–	0	7	–

TABLE 6.17: The Performance of Adaptive Dynamic Heuristics Set (ADHS) Selection Hyper-heuristics: the success rate ($sRate\%$), the average best coloring ($\mu(k_{best})$) and the standard deviation ($\sigma(k_{best})$) of each hyper-heuristic approach on the data clustering problem instances over the 30 runs.

			ADHS-ILTA			ADHS-LACC			ADHS-GDEL		
	Instance	k^*	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$
Synthetic	Square1	4	**100.00**	**4.0**	±0.0	83.33	4.23	±0.57	100.00	4.0	±0.0
	Square4	4	**100.00**	**4.0**	±0.0	76.67	4.33	±0.66	93.33	4.07	±0.25
	Sizes5	4	**80.00**	3.87	±0.43	70.00	**4.0**	±0.64	73.33	4.13	±0.68
	Long1	4	**16.67**	**5.60**	±2.22	3.33	6.0	±2.05	3.33	6.27	±2.24
	Twenty	20	**36.67**	20.93	±1.50	30.00	**20.90**	±1.56	30.00	21.0	±1.49
	Fourty	40	**23.33**	**41.03**	±1.69	6.67	41.6	±1.90	3.33	41.6	±2.08
Gaussian	2D-4c	4	**23.33**	3.87	±0.51	13.33	**4.07**	±0.94	6.67	4.3	±1.39
	2D-10c	10	**20.00**	**8.97**	±1.22	3.33	8.77	±1.43	10.00	8.9	±1.35
	2D-20c	20	**73.33**	18.23	±2.24	43.33	18.57	±2.21	23.33	**18.63**	±2.58
	2D-40c	40	**26.67**	**33.6**	±3.80	10.00	32.57	±3.33	6.67	32.63	±3.37
	10D-4c	4	**20.00**	3.8	±0.961	6.67	**4.03**	±1.35	6.67	4.17	±1.84
	10D-10c	10	**10.00**	8.83	±1.62	0.00	8.93	±1.87	0.00	**9.1**	±1.90
Real	Zoo	7	**56.67**	**7.30**	±1.47	36.67	7.37	±1.63	3.33	7.73	±1.72
	Iris	3	**26.67**	**4.5**	±1.68	3.33	5.23	±1.45	3.33	5.3	±1.32
	Dermatology	6	**26.67**	**6.37**	±1.45	10.00	6.6	±1.57	6.67	6.43	±1.74
	Breast-cancer	2	**13.33**	**3.30**	±0.95	3.33	3.67	±0.99	0.00	3.70	±1.06
	Wins		**16**	**10**	–	0	4	–	0	2	–

TABLE 6.18: The Performance of Simple Random (SR) Selection Hyper-heuristics: the success rate ($sRate\%$), the average best coloring ($\mu(k_{best})$) and the standard deviation ($\sigma(k_{best})$) of each hyper-heuristic approach on the data clustering problem instances over the 30 runs.

			SR-ILTA			SR-LACC			SR-GDEL		
	Instance	k^*	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$	sRate%	$\mu(k_{best})$	$\sigma(k_{best})$
Synthetic	Square1	4	**100.00**	**4.0**	±0.0	80.00	4.27	±0.58	86.66	4.17	±0.46
	Square4	4	**86.67**	**4.13**	±0.35	83.33	4.3	±.75	73.33	4.57	±1.10
	Sizes5	4	**80.00**	**4.13**	±0.63	63.33	4.33	±0.76	70.00	**4.13**	±0.78
	Long1	4	**6.67**	**5.73**	±2.24	**6.67**	5.90	±2.16	**6.67**	5.93	±2.35
	Twenty	20	**26.67**	**20.93**	±1.39	10.00	21.37	±1.69	16.67	21.27	±1.62
	Fourty	40	**3.33**	**41.27**	±1.70	**3.33**	41.67	±1.94	0.00	41.47	±1.82
Gaussian	2D-4c	4	**16.67**	**4.03**	±0.93	3.33	4.3	±1.34	6.67	4.43	±1.38
	2D-10c	10	**10.00**	8.87	±1.28	3.33	8.73	±1.17	6.67	**9.03**	±1.30
	2D-20c	20	**46.67**	18.77	±2.47	26.67	18.93	±2.49	6.67	**19.3**	±2.47
	2D-40c	40	**23.33**	**33.6**	±3.71	3.33	32.6	±3.39	6.67	32.77	±3.51
	10D-4c	4	3.33	3.67	±0.88	0.00	3.6	±1.30	**10.00**	**3.83**	±1.49
	10D-10c	10	0.00	8.77	±1.72	0.00	8.7	±1.73	0.00	**9.03**	±1.73
Real	Zoo	7	**36.67**	**7.33**	±1.56	10.00	7.53	±1.55	6.67	7.90	±1.67
	Iris	3	**20.00**	**5.23**	±1.41	6.67	5.4	±1.28	10.00	5.47	±1.25
	Dermatology	6	**13.33**	**6.60**	±1.61	3.33	6.73	±1.78	3.33	6.73	±1.76
	Breast-cancer	2	**6.67**	**4.03**	±1.45	0.00	4.3	±1.62	0.00	4.27	±1.60
	Wins		**14**	**12**	–	2	0	–	2	5	–

6.6.1 Performance Comparison to Previously Proposed Data Clustering Algorithms

The conclusion which has been drawn in the previous section revealed that the best results were obtained by applying any selection method with ILTA move acceptance. ADHS-ILTA achieved the maximum number of wins in terms of the best success rates in the experimented instances. The performance of ADHS-ILTA is, therefore, taken for further comparison with other known clustering algorithms as shown in Table 6.19 including k-means [MacQueen et al., 1967], Mock [Handl and Knowles, 2007], Ensemble [Hong et al., 2008], Av. Link [Voorhees, 1985], and S. Link [Voorhees, 1985]. For more information about clustring algorithms please regfer to section 3.3.3.

The comparisons shown in Table 6.19 are based on the average number of clusters. The 'wins' row at the bottom of the table indicates the number of winning times each approach achieved over all the Gaussian and the Synthetic instances. Given this table, it is observed that the performance of ADHS-ILTA lies on the top of the other approaches equally with MOCK algorithm scoring five wins, and outperforming the other competing algorithms. ADHS-ILTA scored the best average distinctively in four instances including, Twenty, Fourty, 2D-4c and 2D-20c.

TABLE 6.19: Comparing the performances of different approaches on data clustering problem instances based on the average number of clusters. The entries in bold indicate the best result obtained by the associated algorithm for the given instance.

	Instance	k^*	ADHS-ILTA	k-means	MOCK	Ensemble	Av. link	S. link
Synthetic	Square1	4	**4.00**	**4.00**	**4.00**	**4.00**	**4.00**	2.72
	Square4	4	**4.00**	**4.00**	**4.00**	4.04	4.26	2.00
	Sizes5	4	**3.87**	3.74	**3.87**	3.70	3.76	2.44
	Long1	4	5.60	8.32	8.34	**4.92**	7.78	2.02
	Twenty	20	20.93	–	–	–	–	–
	Fourty	40	41.03	–	–	–	–	–
Gaussian	2D-4c	4	**3.87**	3.69	3.70	2.23	4.50	4.40
	2D-10c	10	8.97	10.66	**9.65**	4.50	15.20	8.00
	2D-20c	20	**18.23**	21.87	17.31	1.24	16.30	16.3
	2D-40c	40	33.60	30.63	**35.26**	2.27	30.90	27.7
	10D-4c	4	3.80	3.59	3.60	5.37	**4.00**	2.00
	10D-10c	10	8.83	**9.02**	8.88	3.86	5.30	2.00
	Wins		**5**	3	**5**	2	2	0

6.7 Summary

In this chapter, the grouping hyper-heuristic framework was applied on problem instances from three different domains, which are graph coloring, timetabling and data

clustering. This investigation is carried out using different selection hyper-heuristics that are formed using all the combinations of the 'simple random', the 'reinforcement learning' and the 'adaptive dynamic heuristic set' selection methods, denoted as {SR, RL, ADHS} respectively, and the 'late acceptance', the 'great deluge' and the 'iteration limited threshold accepting' move acceptance methods, denoted as {LACC, GDEL, ILTA} respectively; generating a total of nine different selection hyper-heuristics. The search heuristics of the framework were implemented using the genetic grouping algorithm encoding (GGAE), which was found to be better than implementation that use other types of grouping solution representations. The problem instances used in this chapter vary in terms of the number of items, the number of groups and the cost functions.

The empirical results show that the proposed framework is indeed sufficiently general and reusable. Also, although the ultimate goal of the grouping framework is not to beat the state of the art techniques that are designed and tuned for specific problems, the results obtained by learning-based hyper-heuristics which uses feedback during the search process turned out to be very competitive when compared to previous approaches from the literature. The following chapter provides final remarks and conclusions on all the work carried out in this research.

Chapter 7

Conclusion

7.1 Context

Heuristics are rule-of-thumb methods acting as alternative approaches for solving computationally hard problems instead of exact methods enabled to exhaustively traverse the search space while solving a given instance. It has been frequently observed that no single problem-specific heuristic has an absolute superiority over another one for solving all instances from a given domain [Burke et al., 2003]. Hyper-heuristics have emerged as high-level search methodologies for solving computationally hard optimization problems by enabling generation or control and mixing of different heuristics (or heuristic components) under a framework yielding new search methodologies. The underlying assumption is that such automated high level search methodologies are more general and reusable approaches that the existing approaches and so they can be applied to the instances with different characteristics from not only a single domain, but also different problem domains[Burke et al., 2003, Ross, 2005].

An important class of combinatorial optimization problems, known as grouping problems, is addressed in this research, aiming at developing a single point-based selection hyper-heuristic framework for all grouping problems. This framework is designed in such a way that it enables the reuse of as many components as possible. Although hyper-heuristics are high level general methodologies, when it comes to designing a hyper-heuristic solver applicable on a specific domain which has not been implemented yet, that requires design and implementation of problem domain level components, such as LLHs. In order to reduce the development effort for a "new" grouping problem domain, this study provides a set of standard LLHs within the proposed framework which formulates the grouping problem as a bi-objective problem.

A software library is implemented based on the framework and the performance of combination of a set of selection hyper-heuristic components are investigated across three problem domains. This library implements all components suggested for the grouping framework based on a carefully chosen representation exploiting previous expertise in the area.

7.2 Summary of Contributions

Considering a bi-objective formulation of grouping problems, this thesis presented the description of a proposed single point-based selection search hyper-heuristic framework that adopts concepts from the multi-objective domain to handle the two grouping objectives. A special acceptance mechanism that enforces the dominance rule throughout the search is described as part of the framework. The framework embeds a total of ten group-oriented, generic search operators/heuristics that can applied to any grouping problem.

This thesis provides the results of some preliminary analysis that were carried out to investigate the performance of the components of the hyper-heuristic framework described above. Firstly, three different implementations of the framework, based on three different solution representations from the literature were compared. A grouping representation that allows the search heuristics to directly affect the groups in a given solution was found to be the best among the representations tested. Then, the effect of the two-phased acceptance algorithm that is part of the described grouping hyper-heuristic framework is analyzed. The empirical results show that the results obtained using this mechanism are superior to those obtained using a traditional selection hyper-heuristics framework.

Also, this thesis provides an extensive investigation of the performance of the grouping hyper-heuristic framework by applying nine different combinations of heuristic selection and acceptance methods implemented using the grouping framework on a total of 47 problem instances from three different problem domains. The results obtained by different approaches are compared against each other, and the best performing approaches are compared against previous problem-specific approaches from the literature. The empirical results show that a learning selection hyper-heuristic developed using the proposed framework turns out to be sufficiently general and reusable. The performance of this hyper-heuristic is either superior to the performances of most of the previously proposed problem-specific approaches, or is highly competitive.

7.3 Extensions and Future Work

The work presented in this thesis is by no means complete. A number of possible improvements and extensions can be made to further enhance the proposed grouping framework.

7.3.1 A smart generic initialization algorithm

The grouping framework presented in this thesis embeds a simple and fast initialization algorithm that constructs the initial solutions in a random fashion. The fundamental reason behind this is that the framework is designed to be generic and applicable to all grouping problems without modification. However, initial solutions for different grouping problems could be constructed in various ways. There are many smart, problem-specific initialization methods that are proposed in the scientific literature [Likas et al., 2003].

The empirical results presented in this thesis show that, despite using such a random initialization approach, the grouping framework is still very competitive with regards to problem-specific approaches. Yet, we acknowledge the fact that, the performance of grouping algorithms is affected to some extent by all initial starting conditions in general [Likas et al., 2003]. Hence, the proposed framework can be enriched by the use of domain specific smart initialization methods. Then again, using such a smart initialization approach is often more time consuming than random initialization. This trade-off is important considering that we run each approach in a time-contract manner, i.e., they terminate after exceeding a time limit. A longer initialization process means less time for the improvement process potentially influencing the performance of the overall approach. Additionally, smart initialization does not automatically indicate improved overall performance. Such methods could generate a locally optimal solution which can not be improved further through the following improvement process.

Better yet, is to design a smart and fast generic initialization algorithm; i.e an algorithm that is not problem-specific, nor random at the same time. This would make a good research direction for the future.

7.3.2 Smart generic low level heuristics

Including additional domain-specific LLHs might lead to a better performance with the proposed framework. Again, the results show that the ten heuristics currently embedded in the framework are generic enough and are capable of finding high quality solutions that are competitive with the state of the art approaches. Of course, a drawback of

extending the LLH set by new domain-specific LLHs is that the framework will no longer be applicable to all grouping domains, requiring some maintenance effort in such a situation. However, hyper-heuristics designed for the framework will be still reusable. Alternatively, the possibility of designing new generic heuristics, that are smarter than the heuristics that are currently embedded in the framework, could be explored.

7.3.3 A population of solutions for each number of groupings

The grouping framework keeps track of a set of non-dominated solutions throughout the search process. However, that set contains only one point for each number of groups in the given range, $k \in [LB, UB]$. Indeed, this does not have to be the case. For a given value of k, a huge number of *different* solutions could have the same cost value; i.e. many solutions in which the items are distributed differently can still have the same number of groups as well as the same cost value. Keeping track of a subset of these solutions as part of the non-dominated set could yield some interesting features and allow the framework to adopt more concepts from the multi-objective optimization optimization domain in particular, as well as the population based approaches in general.

7.4 Final Remarks

The main motivation behind hyper-heuristics is to produce optimization systems that are applicable across different problem domains. Despite being a relatively new area of research compared to other existing approaches such as meta-heuristics, hyper-heuristics have indeed managed to prove their competence as high-level general search methodologies. Yet, there are still many unanswered questions and pending issues that need to be addressed.

Grouping problems comprise a set of complex real world problems. In this thesis, a number of issues related to designing a hyper-heuristics for grouping problems are explored. An attempt to further achieve the original goal of hyper-heuristics beyond the common domain barrier is made by adopting a generic solution representation that allows the same set of LLHs to be used across different grouping problems. Indeed, this work can be enhanced and extended in many various ways.

References

Abramson, D.A., Dang, H., Krisnamoorthy, M., 1999. Simulated annealing cooling schedules for the school timetabling problem. Asia-Pacific Journal of Operational Research 16, 1–22.

Abuhamdah, A., 2010. Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. International Journal of Computer Science and Network Security (IJCSNS) 10, 192–200.

Acuña, A., Parada, V., Gatica, G., 2011. Cross-domain heuristic search challenge: GISS algorithm presentation. http://www.asap.cs.nott.ac.uk/external/chesc2011/.

Agustın-Blas, L.E., Salcedo-Sanz, S., Jiménez-Fernández, S., Carro-Calvo, L., Del Ser, J., Portilla-Figueras, J.A., 2012. A new grouping genetic algorithm for clustering problems. Expert Systems with Applications 39, 9695–9703. URL: http://dx.doi.org/10.1016/j.eswa.2012.02.149, doi:10.1016/j.eswa.2012.02.149.

Ankerst, M., Breunig, M.M., peter Kriegel, H., Sander, J., 1999. Optics: Ordering points to identify the clustering structure, ACM Press. pp. 49–60.

Auger, A., Bader, J., Brockhoff, D., Zitzler, E., 2009. Theory of the hypervolume indicator: Optimal u-distributions and the choice of the reference point, in: Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms, ACM, New York, NY, USA. pp. 87–102. URL: http://doi.acm.org/10.1145/1527125.1527138, doi:10.1145/1527125.1527138.

Avanthay, C., Hertz, A., Zufferey, N., 2003. A variable neighborhood search for graph coloring. European Journal of Operational Research 151, 379–388.

Bache, K., Lichman, M., 2013. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Science. URL: http://archive.ics.uci.edu/ml.

Bader-El-Den, M., Poli, R., 2007. Generating sat local-search heuristics using a gp hyper-heuristic framework, in: Monmarché, N., Talbi, E.G., Collet, P., Schoenauer,

M., Lutton, E. (Eds.), Evolution Artificielle, 8th International Conference, Springer, Tours, France. pp. 37–49.

Bai, R., 2005. An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation. Ph.D. thesis. Computer Science and Information Technology, University of Nottingham.

Bai, R., Kendall, G., 2005. An investigation of automated planograms using a simulated annealing based hyper-heuristics, in: Ibaraki, T., Nonobe, K., Yagiura, M. (Eds.), Metaheuristics: Progress as Real Problem Solver - (Operations Research/Computer Science Interface Serices, Vol.32). Springer, pp. 87–108.

Bilgin, B., Özcan, E., Korkmaz, E.E., 2007. An experimental study on hyper-heuristics and exam timetabling, in: Proceedings of the 6th international conference on Practice and theory of automated timetabling VI, Springer-Verlag, Berlin, Heidelberg. pp. 394–412.

Brown, C.E., Sumichrast, R.T., 2003. Impact of the replacement heuristic in a grouping genetic algorithm. Computers & OR 30, 1575–1593.

Brownlee, A.E.I., Swan, J., Özcan, E., Parkes, A.J., 2014. Hyperion2: a toolkit for {meta-, hyper-} heuristic research, in: Arnold, D.V., Alba, E. (Eds.), Genetic and Evolutionary Computation Conference, GECCO '14, Companion Material Proceedings, ACM. pp. 1133–1140.

Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J., 2009a. Hyflex: A flexible framework for the design and analysis of hyper-heuristics., in: Proceedings of the Multidisciplinary International Scheduling Conference (MISTA09), pp. 790–797.

Burke, E., Gendreau, M., Hyde, M., Kendall, G., McCollum, B., Ochoa, G., Parkes, A., Petrovic, S., 2011. The cross-domain heuristic search challenge - an international research competition, in: Yao, X., Coello, C.A.C. (Eds.), Proceedings of Learning and Intelligent Optmization (LION5), pp. 631–634.

Burke, E., Hyde, M., Kendall, G., 2006a. Evolving bin packing heuristics with genetic programming, in: Runarsson, T., Beyer, H.G., Burke, E., Guervos, G.M., Juan, J., Whitley, D., Yao, X. (Eds.), Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), Reykjavik, Iceland. pp. 860–869. doi:10.1007/11844297_87.

Burke, E.K., Bykov, Y., 2008. A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems, in: PATAT '08 Proceedings of the 7th International Conference

on the Practice and Theory of Automated Timetabling. URL: http://w1.cirrelt.ca/~{}patat2008/PATAT_7_PROCEEDINGS/Papers/Bykov-HC2a.pdf.

Burke, E.K., Gendreau, M., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. JORS 64, 1695–1724.

Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S., 2003. Hyper-heuristics: An emerging direction in modern search technology, in: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics. Kluwer, pp. 457–474.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J., 2009b. A classification of hyper-heuristic approaches, in: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer. International Series in Operations Research & Management Science.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J., 2009c. Exploring hyper-heuristic methodologies with genetic programming , 177–201.

Burke, E.K., Hyde, M., Kendall, G., Woodward, J., 2007a. Automatic heuristic generation with genetic programming: Evolving a jack-of-alltrades or a master of one, in: Genetic and Evolutionary Computation Confrence, GECCO 2007, Proceedings, pp. 7–11.

Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R., 2007b. The scalability of evolved on line bin packing heuristics, in: Srinivasan, D., Wang, L. (Eds.), 2007 IEEE Congress on Evolutionary Computation, IEEE Computational Intelligence Society. IEEE Press, Singapore. pp. 2530–2537.

Burke, E.K., Kendall, G., sır, M.M., Özcan, E., 2008. Monte carlo hyper-heuristics for examination timetabling, in: Conference on the Practice and Theory of Automated Timetabling (PATAT 2008).

Burke, E.K., Landa-Silva, J.D., Soubeiga, E., 2005. Multi-objective hyper-heuristic approaches for space allocation and timetabling, in: Ibaraki, T., Nonobe, K., Yagiura, M. (Eds.), Meta-heuristics: Progress as Real Problem Solvers. Springer. 5th Metaheuristics International Conference (MIC 2003), pp. 129–158.

Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R., 2007c. A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research 176, 177–192. URL: http://www.asap.cs.nott.ac.uk/publications/pdf/EJOR.pdf.

Burke, E.K., Newall, J.P., 1998. A multi-stage evolutionary algorithm for the timetable problem. IEEE Transactions on Evolutionary Computation .

Burke, E.K., Petrovic, S., Qu, R., 2006b. Case-based heuristic selection for timetabling problems. J. of Scheduling 9, 115–132. URL: http://dx.doi.org/10.1007/s10951-006-6775-y, doi:10.1007/s10951-006-6775-y.

Burke, E.K., Soubeiga, E., 2003. Scheduling nurses using a tabu-search hyperheuristic, in: Proc. of the 1st MISTA, pp. 197–218.

Caramia, M., Dell'Olmo, P., Italiano, G.F., 2001. New algorithms for examination timetabling, in: Algorithm Engineering 4th International Workshop 2000, Springer-Verlag, Berlin Heidelberg New York. pp. 230–241.

Carter, M.W., 1986. A survey of practical applications of examination timetabling algorithms. Oper. Res. 34, 193–202. URL: http://portal.acm.org/citation.cfm?id=11117.11118, doi:10.1287/opre.34.2.193.

Carter, M.W., Laporte, G., 1996. Recent developments in practical examination timetabling, in: Practice and Theory of Automated Timetabling, pp. 3–21.

Carter, M.W., Laporte, G., Lee, S.T., 1996. Examination timetabling: algorithmic strategies and applications. Journal of the Operational Research Society 47, 373–383.

Ceschia, S., Di Gaspero, L., Schaerf, A., 2011. Tabu search techniques for the heterogeneous vehicle routing problem with time windows and carrier-dependent costs. Journal of Scheduling 14, 601–615.

Ceschia, S., Schaerf, A., Stützle, T., 2013. Local search techniques for a routing-packing problem. Computers and Industrial Engineering 66, 1138–1149.

Chakhlevitch, K., Cowling, P.I., 2008. Hyperheuristics: Recent developments, in: Cotta, C., Sevaux, M., Sörensen, K. (Eds.), Adaptive and Multilevel Metaheuristics. Springer. volume 136 of *Studies in Computational Intelligence*, pp. 3–29.

Chan, C.Y., Xue, F., Ip, W.H., Cheung, C.F., 2012. A hyper-heuristic inspired by pearl hunting, in: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. Lecture Notes in Computer Science, pp. 349–353.

Chang, D.X., Zhang, X.D., Zheng, C.W., 2009. A genetic algorithm with gene rearrangement for k-means clustering. Pattern Recognition 42, 1210 – 1222. URL: http://www.sciencedirect.com/science/article/pii/S0031320308004779, doi:http://dx.doi.org/10.1016/j.patcog.2008.11.006.

Cichowicz, T., Drozdowski, M., Frankiewicz, M., Pawlak, G., Rytwinski, F., Wasilewski, J., 2012. Five phase and genetic hive hyper-heuristics for the cross-domain search, in: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. Lecture Notes in Computer Science, pp. 354–359.

Coello, C.C., Lamont, G., van Veldhuizen, D., 2007. Evolutionary Algorithms for Solving Multi-Objective Problems. Genetic and Evolutionary Computation. 2nd ed., Springer, Berlin, Heidelberg.

Coffman, Jr., E.G., Garey, M.R., Johnson, D.S., 1997. Approximation algorithms for np-hard problems, PWS Publishing Co., Boston, MA, USA. chapter Approximation algorithms for bin packing: a survey, pp. 46–93.

Cowling, P., Chakhlevitch, K., 2003. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC2003), IEEE Computer Society Press, Canberra, Australia. pp. 1214–1221.

Cowling, P., Kendall, G., Han, L., 2002. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem, in: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02, IEEE Computer Society, Washington, DC, USA. pp. 1185–1190. URL: `http://portal.acm.org/citation.cfm?id=1251972.1252373`.

Cowling, P., Kendall, G., Soubeiga, E., 2000. A hyperheuristic approach to scheduling a sales summit, in: Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000, Springer, Konstanz, Germany. pp. 176–190.

Cowling, P., Kendall, G., Soubeiga, E., 2001. A parameter-free hyperheuristic for scheduling a sales summit, in: Proceedings of the 4th Metahuristics International Conference (MIC 2001), Porto Portugal. pp. 127–131. URL: `http://www.cs.nott.ac.uk/~gxk/papers/mic01exs.pdf`.

Davies, D.L., Bouldin, D.W., 1979. A cluster separation measure. IEEE Trans. Pattern Anal. Mach. Intell. 1, 224–227. URL: `http://dx.doi.org/10.1109/TPAMI.1979.4766909`, doi:`10.1109/TPAMI.1979.4766909`.

Davis, L.D., Mitchell, M., 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold.

Dempster, A.P., Laird, N.M., Rubin, D.B., 1977. Maximum likelihood from incomplete data via the em algorithm. JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B 39, 1–38.

Deng, S., He, Z., Xu, X., 2010. G-ANMI: A mutual information based genetic clustering algorithm for categorical data. Knowl.-Based Syst. 23, 144–149. URL: `http://dblp.uni-trier.de/db/journals/kbs/kbs23.html#DengHX10`.

Di Gaspero, L., Urli, T., 2012. Evaluation of a family of reinforcement learning cross-domain optimization heuristics, in: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. Lecture Notes in Computer Science, pp. 384–389.

Ding, H., El-Keib, A., Smith, R., 1994. Optimal clustering of power networks using genetic algorithms. Electric Power Systems Research 30, 209–214.

Dowsland, K.A., 1998. Nurse scheduling with tabu search and strategic oscillation. European Journal of Operational Research 106, 393–407.

Du, J., Korkmaz, E.E., Alhajj, R., Barker, K., 2004. Novel clustering that employs genetic algorithm with new representation scheme and multiple objectives., in: Kambayashi, Y., Mohania, M.K., Wöß, W. (Eds.), Data Warehousing and Knowledge Discovery, 6th International Conference, DaWaK 2004, Zaragoza, Spain, September 1-3, 2004, Proceedings, Springer. pp. 219–228.

Duda, R.O., Hart, P.E., Stork, D.G., 2001. Pattern Classification. 2 ed., Wiley-Interscience.

Dueck, G., 1993. New optimization heuristics: The great deluge algorithm and the record-to-record travel. Journal of Computational Physics 104, 86–92. doi:10.1006/jcph.1993.1010.

Elomari, J., 2011. Self-search. http://www.asap.cs.nott.ac.uk/external/chesc2011/.

Falkenauer, E., 1992. The grouping genetic algorithms: Widening the scope of the GAs. Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL) 33, 79–102.

Falkenauer, E., 1996. A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 5–30.

Falkenauer, E., 1998. Genetic Algorithms and Grouping Problems. John Wiley & Sons, Inc., New York, NY, USA.

lan Fang, H., Ross, P., Corne, D., 1994. A promising hybrid ga heuristic approach for open-shop scheduling problems, in: ECAI 94 Proceedings of the 11th European Conference on Artificial Intelligence, John Wiley & Sons, Ltd. pp. 590–594.

Fisher, H., Thompson, G.L., 1963. Probabilistic learning combinations of local job-shop scheduling rules .

Fukunaga, A.S., 2008. Automated discovery of local search heuristics for satisfiability testing. Evol. Comput. 16, 31–61. URL: http://dx.doi.org/10.1162/evco.2008.16.1.31, doi:http://dx.doi.org/10.1162/evco.2008.16.1.31.

Galinier, P., Hao, J.K., 1999. Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. 3, 379–397.

Gamboa, D., Rego, C., Glover, F., 2006. Implementation analysis of efficient heuristic algorithms for the traveling salesman problem. Comput. Oper. Res. 33, 1154–1172. URL: http://portal.acm.org/citation.cfm?id=1119551.1119568, doi:10.1016/j.cor.2005.06.014.

García-Villoria, A., Salhi, S., Corominas, A., Pastor, R., 2011. Hyper-heuristic approaches for the response time variability problem. European Journal of Operational Research 211, 160–169.

Gaw, A., Rattadilok, P., Kwan, R.S.K., 2004. Distributed choice function hyperheuristics for timetabling and scheduling, in: Practice and Theory of Automated Timetabling V, Springer Lecture notes in Computer Science. Volume 3616. (2005), pp. 51–70.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13, 533–549.

Goldberg, D., 1989. Genetic algorithms in search, optimization and machine learning. Addison-Wesley.

Gomez, J., 2004. Self adaptation of operator rates in evolutionary algorithms, in: Deb, K. (Ed.), Genetic and Evolutionary Computation (GECCO '04). Springer Berlin Heidelberg. volume 3102 of *Lecture Notes in Computer Science*, pp. 1162–1173.

Gutin, G., Karapetyan, D., 2009. A memetic algorithm for the generalized traveling salesman problem. Natural Computing 9, 47–60.

Halkidi, M., Batistakis, Y., Vazirgiannis, M., 2001. On clustering validation techniques. Journal of Intelligent Information Systems 17, 107–145.

Hancock, P., 1992. Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification, in: Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on, pp. 108–122. doi:10.1109/COGANN.1992.273944.

Handl, J., Knowles, J.D., 2007. An evolutionary approach to multiobjective clustering. IEEE Trans. Evolutionary Computation 11, 56–76.

Hart, E., Ross, P., Nelson, J., 1998. Solving a real-world problem using an evolving heuristically driven schedule builder. Evol. Comput. 6, 61–80. URL: http://dx.doi.org/10.1162/evco.1998.6.1.61, doi:http://dx.doi.org/10.1162/evco.1998.6.1.61.

Hart, E., Ross, P., Nelson, J., 1999. Scheduling chicken catching - an investigationinto the success of a genetic algorithm on a real world scheduling problem. Annals of Operations Research 92, 363–380. URL: http://dx.doi.org/10.1023/A:1018951218434. 10.1023/A:1018951218434.

Hertz, A., Werra, D.D., 1987. Using tabu search techniques for graph coloring. Computing 39, 345–351. URL: http://dx.doi.org/10.1007/BF02239976, doi:10.1007/BF02239976.

Holland, J.H., 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge, MA, USA.

Hong, Y., Kwong, S., Chang, Y., Ren, Q., 2008. Unsupervised feature selection using clustering ensembles and population based incremental learning algorithm. Pattern Recognition 41, 2742–2756. URL: http://dx.doi.org/10.1016/j.patcog.2008.03.007, doi:10.1016/j.patcog.2008.03.007.

Hruschka, E.R., Ebecken, N.F.f., 2003. A genetic algorithm for cluster analysis. Intell. Data Anal. 7, 15–25. URL: http://dl.acm.org/citation.cfm?id=1293920.1293922.

Hsiao, P.C., Chiang, T.C., Fu, L.C., 2012. A VNS-based hyper-heuristic with adaptive computational budget of local search, in: IEEE Congress on Evolutionary Computation (CEC '12), pp. 1–8.

Hurink, J., Knust, S., 2005. Tabu search algorithms for job-shop problems with a single transport robot. European Journal of Operational Research 162, 99–111. Logistics: From Theory to Application.

Hyde, M., 2010. A Genetic Programming Hyper-Heuristic Approach to Automated Packing. Ph.D. thesis. School of Computer Science, The University of Nottingham.

Jain, A.K., Dubes, R.C., 1988. Algorithms for clustering data. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Jain, A.K., Murty, M.N., Flynn, P.J., 1999. Data clustering: a review. ACM Computing Surveys 31, 264–323.

Jensen, T.R., Toft, B., 1994. Graph Coloring Problems. 1 edition.

Johnson, D.J., Trick, M.A. (Eds.), 1996. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993. American Mathematical Society, Boston, MA, USA.

Johnson, D.S., Aragon, C.R., Mcgeoctt, L.A., Schevon, C., 1991. Optimization by simulated annealing: an experimental evaluation; part ii. Operational Research .

Johnston, M., Liddle, T., Miller, J., Zhang, M., 2011. A hyperheuristic based on dynamic iterated local search. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`.

Kashef, R., Kamel, M.S., 2010. Cooperative clustering. Pattern Recognition 43, 2315–2329.

Keller, R.E., Poli, R., 2008. Cost-benefit investigation of a genetic-programming hyperheuristic, in: Proceedings of the Evolution artificielle, 8th international conference on Artificial evolution, Springer-Verlag, Berlin, Heidelberg. pp. 13–24. URL: `http://portal.acm.org/citation.cfm?id=1793671.1793674`.

Kendall, G., Hussin, N., 2005. An investigation of a tabu search based hyperheuristic for examination timetabling, in: Kendall, G., Burke, E., Petrovic, S., Gendreau, M. (Eds.), Selected papers from MISTA 2003, Springer. pp. 309–328. URL: `http://www.mistaconference.org`. an extended abstract of this paper appeared in the Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003) pp 226-233.

Kendall, G., Hussin, N.M., 2004. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology, in: Practice and Theory of Automated Timetabling, pp. 270–293.

Kendall, G., Mohamad, M., 2004. Channel assignment optimisation using a hyperheuristic, in: Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent Systems (CIS2004), Singapore. pp. 790–795.

Khamassi, I., 2011. Ant-Q hyper heuristic approach applied to the cross-domain heuristic search challenge problems. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`.

Kirovski, D., Potkonjak, M., 1998. Efficient coloring of a large spectrum of graphs, in: 35th Design Automation Conference Proceedings, pp. 427–432.

Kohonen, T., 1990. The self-organizing map. Proceedings of the IEEE 78, 1464–1480.

Korkmaz, E., 2006. A two-level clustering method using linear linkage encoding, in: Runarsson, T., Beyer, H.G., Burke, E.K., Merelo-Guervós, J., Whitley, L., Yao, X.

(Eds.), Parallel Problem Solving from Nature - PPSN IX. Springer Berlin Heidelberg. volume 4193 of *Lecture Notes in Computer Science*, pp. 681–690. URL: http://dx.doi.org/10.1007/11844297_69, doi:10.1007/11844297_69.

Korkmaz, E.E., 2010. Multi-objective genetic algorithms for grouping problems. Appl. Intell. 33, 179–192.

Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G., 2003. Genetic programming iv: Routine human-competitive machine intelligence by means of genetic programming.

Kubalík, J., 2012. Hyper-heuristic based on iterated local search driven by evolutionary algorithm, in: Hao, J.K., Middendorf, M. (Eds.), Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg. volume 7245 of *Lecture Notes in Computer Science*, pp. 148–159.

Larose, M., 2011. A hyper-heuristic for the CHeSC 2011, in: The 53rd Annual Conference of the UK Operational Research Society (OR53).

von Laszewski, G., 1991. Intelligent structural operators for the k-way graph partitioning problem, in: Belew, R.K., Booker, L.B. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Diego, CA. pp. 45–52. URL: http://dblp.uni-trier.de/db/conf/icga/icga1991.html#Laszewski91.

Lehrbaum, A., Musliu, N., 2012. A new hyperheuristic algorithm for cross-domain search problems, in: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. Lecture Notes in Computer Science, pp. 437–442.

Leighton, F.T., 1979. A graph coloring algorithm for large scheduling problems, in: Journal of Reasearch of the National Bureau of Standards, pp. 489–506.

Likas, A., Vlassis, N., Verbeek, J.J., 2003. The global k-means clustering algorithm. Pattern Recognition 36, 451 – 461. URL: http://www.sciencedirect.com/science/article/pii/S0031320302000602, doi:http://dx.doi.org/10.1016/S0031-3203(02)00060-2. biometrics.

Liu, Y., Wu, X., Shen, Y.D., 2011. Automatic clustering using genetic algorithms. Applied Mathematics and Computation 218, 1267–1279. URL: http://dblp.uni-trier.de/db/journals/amc/amc218.html#LiuWS11.

Maashi, M., zcan, E., Kendall, G., 2014. A multi-objective hyper-heuristic based on choice function. Expert Systems with Applications 41, 4475 – 4493. URL: http:

//www.sciencedirect.com/science/article/pii/S095741741400013X, doi:http://dx.doi.org/10.1016/j.eswa.2013.12.050.

MacQueen, J., et al., 1967. Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, California, USA. pp. 281–297.

Madeira, S.C., Oliveira, A.L., 2004. Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Transactions on Computational Biology and Bioinformatics 1, 24–45.

Mascia, F., Stützle, T., 2012. A non-adaptive stochastic local search algorithm for the CHeSC 2011 competition, in: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. Lecture Notes in Computer Science, pp. 101–114.

McClymont, K., Keedwell, E., 2011a. A single objective variant of the online selective Markov chain hyper-heuristic (MCHH-S). http://www.asap.cs.nott.ac.uk/external/chesc2011/.

McClymont, K., Keedwell, E.C., 2011b. Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11), ACM, New York, NY, USA. pp. 2003–2010.

Meignan, D., 2011. An evolutionary programming hyper-heuristic with co-evolution for CHeSCÍ1, in: The 53rd Annual Conference of the UK Operational Research Society (OR53).

Merlot, L.T.G., Boland, N., Hughes, B.D., Stuckey, P.J., 2002. A hybrid algorithm for the examination timetabling problem., in: Burke, E. K.; De Causmaecker, P. (Ed.), Proceedings of Practice and Theory of Automated Timetabling, Fourth International Conference, Gent, Belgium. pp. 348–371.

Misir, M., Verbeeck, K., Causmaecker, P.D., Berghe, G.V., 2013. A new hyper-heuristic as a general problem solver: an implementation in hyflex. J. Scheduling 16, 291–311.

Mitra, S., Banka, H., 2006. Multi-objective evolutionary biclustering of gene expression data. Pattern Recognition 39, 2464–2477.

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical Report Technical Report C3P Report 826. Caltech Concurrent Computation Program, California Institute of Technology.

Nareyek, A., 2003. Choosing search heuristics by non-stationary reinforcement learning, in: Resende, M.G.C., de Sousa, J.P. (Eds.), Metaheuristics: Computer Decision-Making, Kluwer. pp. 523–544.

Nareyek, A., 2004. Choosing search heuristics by non-stationary reinforcement learning, in: Resende, M.G.C., de Sousa, J.P., Viana, A. (Eds.), Metaheuristics. Kluwer Academic Publishers, Norwell, MA, USA, pp. 523–544. URL: `http://dl.acm.org/citation.cfm?id=982409.982435`.

Norenkov, I., 1994. Scheduling and allocation for simulation and synthesis of cad system hardware, pp. 20–24.

Núñez, J.L., Ceballos, A., 2011. A general purpose hyper-heuristic based on ant colony optimization. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`.

Oltean, M., 2005. Evolving evolutionary algorithms using linear genetic programming. Evol. Comput. 13, 387–410. URL: `http://dx.doi.org/10.1162/1063656054794815`, doi:`http://dx.doi.org/10.1162/1063656054794815`.

Özcan, E., 2007. Memes, self-generation and nurse rostering, in: Burke, E.K., Rudova, H. (Eds.), Practice and Theory of Automated Timetabling VI. Springer Berlin Heidelberg. volume 3867 of *Lecture Notes in Computer Science*, pp. 85–104.

Özcan, E., Bilgin, B., Korkmaz, E., 2008. A comprehensive analysis of hyperheuristics. Intelligent Data Analysis 12, 1–21.

Özcan, E., Bykov, Y., Birben, M., Burke, E.K., 2009. Examination timetabling using late acceptance hyper-heuristics, in: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2009), pp. 997–1004.

Paquete, F.L., Fortseca, C.M., Pt, E.L., . A study of examination timetabling with multiobjective evolutionary algorithms, in: Proc. of the 4th Metaheuristics International Conference (MIC 2001.

Park, Y.J., Song, M.S., 1998. A genetic algorithm for clustering problems, in: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (Eds.), Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, University of Wisconsin, Madison, Wisconsin, USA. pp. 568–575.

Qu, R., Burke, E.K., McCollum, B., Merlot, L., Lee, S., 2009. A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12, 55–89.

Radcliffe, N.J., 1991. Formal analysis and random respectful recombination, in: Proceedings of the 4th International Conference on Genetic Algorithm, pp. 222–229.

Radcliffe, N.J., Surry, P.D., 1994. Fitness variance of formae and performance prediction., in: Whitley, L.D., Vose, M.D. (Eds.), FOGA, Morgan Kaufmann Publishers Inc.. pp. 51–72. URL: http://dblp.uni-trier.de/db/conf/foga/foga1994.html#RadcliffeS94.

Rand, W., 1971. Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association 66, 846–850.

Reeves, C., 1993. Hybrid genetic algorithms for bin-packing and related problems. Annals of Operations Research 63, 371–396.

Ross, P., 2005. Hyper-heuristics, in: Burke, E.K., Kendall, G. (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer. chapter 17, pp. 529–556.

Ross, P., Schulenberg, S., Marin-Blazquez, J.G., Hart, E., 2002. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. , 942–948.

Rousseeuw, P., 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20, 53–65. URL: http://dx.doi.org/10.1016/0377-0427(87)90125-7, doi:10.1016/0377-0427(87)90125-7.

Ryser-Welch, P., Miller, J.F., 2014. A review of hyper-heuristic frameworks, in: Proceedings of the Evo20 Workshop, AISB 2014.

Saha, S., Kumar, R., Baboo, G., 2013. Characterization of graph properties for improved pareto fronts using heuristics and {EA} for bi-objective graph coloring problem. Applied Soft Computing 13, 2812 – 2822. URL: http://www.sciencedirect.com/science/article/pii/S1568494612003018, doi:http://dx.doi.org/10.1016/j.asoc.2012.06.021.

Sastry, K., Goldberg, D.E., Kendall, G., 2014. Genetic algorithms, in: Burke, E.K., Kendall, G. (Eds.), Search Methodologies. Springer US, pp. 93–117.

Sim, K., 2011. KSATS-HH: a simulated annealing hyper-heuristic with reinforcement learning and tabu-search. http://www.asap.cs.nott.ac.uk/external/chesc2011/.

Smith, D., 1985. Bin packing with adaptive search, in: Proceedings of the 1st International Conference on Genetic Algorithms, L. Erlbaum Associates Inc., Hillsdale, NJ, USA. pp. 202–207.

Sörensen, K., Glover, F.W., 2013. Metaheuristics, in: Gass, S.I., Fu, M.C. (Eds.), Encyclopedia of Operations Research and Management Science. Springer US, pp. 960–970.

Soubeiga, E., 2003. Development and Application of Hyperheuristics to Personnel Scheduling. Ph.D. thesis. Computer Science and Information Technology, University of Nottingham.

Swan, J., Özcan, E., Kendall, G., 2011. Hyperion - a recursive hyper-heuristic framework, in: Coello, C.A.C. (Ed.), Learning and Intelligent Optimization. Springer Berlin Heidelberg. volume 6683 of *Lecture Notes in Computer Science*, pp. 616–630.

Talbi, E.G., Bessiere, P., 1991. A parallel genetic algorithm for the graph partitioning problem, in: Proceedings of the 5th international conference on Supercomputing, ACM. pp. 312–320.

Terashima-Marín, H., Ross, P., Valenzuela-Rendón, M., 1999. Evolution of constraint satisfaction strategies in examination timetabling, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), Morgan Kaufmann, San Francisco. pp. 635–642.

Tucker, A., Crampton, J., Swift, S., 2005. Rgfga: An efficient representation and crossover for grouping genetic algorithms. Evolutionary Computation 13, 477–499. URL: `http://dblp.uni-trier.de/db/journals/ec/ec13.html#TuckerCS05`.

Ülker, O., Korkmaz, E.E., Özcan, E., 2008. A grouping genetic algorithm using linear linkage encoding for bin packing, in: Parallel Problem Solving from Nature, pp. 1140–1149.

Ülker, Ö., Özcan, E., Korkmaz, E.E., 2006. Linear linkage encoding in grouping problems: Applications on graph coloring and timetabling, in: Burke, E.K., Rudová, H. (Eds.), PATAT, Springer. pp. 347–363.

Van Driessche, R., Piessens, R., 1992. Load balancing with genetic algorithms, in: Männer, R., Manderick, B. (Eds.), Parallel Problem Solving from Nature, 2, pp. 341–350. URL: `https://lirias.kuleuven.be/handle/123456789/133983`.

Voorhees, E.M., 1985. The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval. Ph.D. thesis.

Wu, Q., Hao, J.K., 2012. An effective heuristic algorithm for sum coloring of graphs. Computers & OR 39, 1593–1600.

Xiao, J., Yan, Y., Zhang, J., Tang, Y., 2010. A quantum-inspired genetic algorithm for k-means clustering. Expert Systems with Applications 37, 4966–4973.

Yılmaz, B., Korkmaz, E.E., 2010. Representation issue in graph coloring, in: ISDA, IEEE. pp. 1171–1176.

Zahraie, B., Roozbahani, A., 2011. {SST} clustering for winter precipitation prediction in southeast of iran: Comparison between modified k-means and genetic algorithm-based clustering methods. Expert Systems with Applications 38, 5919 – 5929. URL: http://www.sciencedirect.com/science/article/pii/S0957417410012686, doi:http://dx.doi.org/10.1016/j.eswa.2010.11.031.

Zitzler, E., Thiele, L., 1998. Multiobjective optimization using evolutionary algorithms - a comparative case study, in: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.P. (Eds.), PPSN, Springer. pp. 292–304.