

HYPER-HEURISTICS FOR GROUPING PROBLEMS

by

Murat Birben

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of

Master of Science

in

Computer Engineering

Yeditepe University

2011

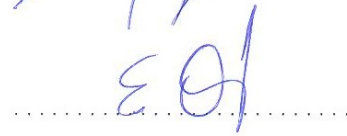
HYPER-HEURISTICS FOR GROUPING PROBLEMS

APPROVED BY:

Assist. Prof. Dr. Ender ÖZCAN
(Thesis Supervisor)



Assist. Prof. Dr. Esin ONBAŞIOĞLU



Assist. Prof. Dr. Şima ETANER UYAR
(Istanbul Technical University)



DATE OF APPROVAL: .../.../2011

ACKNOWLEDGEMENTS

I would like to thank Dr.Ender Özcan for his unbelievable support and mentorship during the preparation of my thesis. It was a great opportunity and honor for me to work with him. This thesis would not have been possible unless the advices, helps and support of him.

I would also like to thank Assist.Prof.Dr.Esin Onbaşıođlu and Assist.Prof.Dr.Şima Etaner Uyar for serving on my thesis committee.

I am indebted to department secretary of department of computer engineering Suna Dündar for her encouragement and standing by me whenever i need.

I am also indebted to Mehmet Aksayan and my ARDIC family for their patience and support during my thesis.

Last but not least, I would like to express my sincere gratitude to my mother, my father, my sister and my grandmother. Without their endless patience and support, I would not be able to achieve anything in my life.

ABSTRACT

HYPER-HEURISTICS FOR GROUPING PROBLEMS

Hyper-heuristics emerge as domain independent methodologies to solve hard computational search problems by performing search over the heuristics rather than directly solutions. One of the main goals of hyper-heuristic research is to support and investigate into the development of more general approaches applicable across different problem domains. Grouping problems requires partitioning of a set of items into mutually disjoint subsets subject to constraints. In this study, high level selection hyper-heuristics are investigated embedding a set of low level heuristics for grouping problems based on an efficient representation, referred to as linear linkage encoding. The empirical results over multi-objective and single objective grouping problems, such as graph coloring, examination timetabling, data clustering and bin packing show that the proposed grouping hyper-heuristic framework is sufficiently general providing high quality solutions at each domain.

ÖZET

GRUPLAMA PROBLEMLERİ İÇİN ÇOK HEDEFLİ ÜST BULUŞSALLAR

Üst sezgiseller, çözümü zor hesaplamaya dayalı arama problemlerini direk çözümler üzerinde arama yapmak yerine sezgiseller üzerinde arama yaparak çözmeye çalışan etki alanı bağımsız bir metodoloji olarak ortaya çıkmaktadır. Üst sezgisel araştırmanın ana hedefi, farklı problem alanlarında uygulanabilir genel yaklaşımların geliştirilmesini desteklemek ve araştırmaktır. Çok amaçlı optimizasyon birden fazla ve genellikle çelişen amacı optimize etmeyi hedefler. Bu çalışmada, yüksek düzey üst sezgiseller gruplama problemleri için bir grup düşük düzey sezgiseller kullanarak doğrusal bağlantı kodlaması olarak adlandırılan verimli bir sunuma bağlı olarak araştırıldı. Çok amaçlı ve tek amaçlı, çizge boyama, sınav çizelgeleme, veri gruplama ve sele doldurma problemleri gibi gruplama problemleri üzerindeki deneysel sonuçlarda, önerilen gruplama üst sezgisel sistemi bütün etki alanlarında yeterince genel yüksek kalitede sonuçlar vermiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iii
ABSTRACT	iv
ÖZET.....	v
TABLE OF CONTENTS	vi
LIST OF FIGURES	ix
LIST OF TABLES	xii
LIST OF SYMBOLS/ABBREVIATIONS	xiv
1. INTRODUCTION	1
2. BACKGROUND	3
2.1. HYPER-HEURISTICS	3
2.1.1. The Hyper-heuristic Concept	3
2.1.2. Classification of Hyper-heuristics	4
2.1.3. Selection Hyper-heuristics: Heuristic Selection and Move Acceptance Methods.....	6
2.1.4. Performance Comparison Studies	15
2.1.5. Hyper-heuristic Tools	17
2.2. MULTI-OBJECTIVE OPTIMIZATION.....	17
3. GROUPING PROBLEMS	21
3.1. LINEAR LINKAGE ENCODING	22
3.2. ELBOW CRITERION	24
3.3. DATA CLUSTERING	25
3.4. EXAM TIMETABLING.....	26
3.5. GRAPH COLORING.....	27
3.6. BIN PACKING.....	28
4. SELECTION HYPERHEURISTICS FOR GROUPING PROBLEMS	30
4.1. FITNESS EVALUATION	30
4.1.1. Graph Coloring and Exam Timetabling.....	30
4.1.2. Data Clustering.....	31
4.1.3. Bin Packing.....	31

4.1.4.	Delta Fitness Evaluation	32
4.2.	HEURISTIC SELECTION METHODS	32
4.2.1.	Simple Random	33
4.2.2.	Reinforcement Learning	33
4.2.3.	Modified Reinforcement Learning	35
4.3.	MOVE ACCEPTANCE METHODS	36
4.3.1.	Great Deluge	37
4.3.2.	Improve or Equal	38
4.3.3.	Late Acceptance	38
4.4.	LOW LEVEL HEURISTICS	40
4.4.1.	Swap	41
4.4.2.	Merge	42
4.4.3.	Merge Tournament	43
4.4.4.	Merge Most Conflicting	44
4.4.5.	Divide	46
4.4.6.	Divide Tournament	48
4.4.7.	Divide Most Conflicting	50
4.4.8.	Change	52
4.4.9.	Change from Most Conflicting to Most Suitable	52
4.4.10.	Change with Most Suitable	55
4.4.11.	Low-level Heuristics for Bin Packing Problem	58
4.5.	TYPES OF MULTI-OBJECTIVE HYPER-HEURISTIC FRAMEWORKS ...	59
4.5.1.	Generic Hyper-heuristic	60
4.5.2.	Cyclic Candidate Solution Selection Hyper-heuristic	60
4.5.3.	Apply to All Candidate Solutions Hyper-heuristic	62
4.6.	SINGLE OBJECTIVE HYPER-HEURISTICS	62
5.	EXPERIMENTS	67
5.1.	EXPERIMENTAL DATA	67
5.2.	PARAMETER TUNING EXPERIMENTS	71
5.2.1.	Tuning Tournament Size	72
5.2.2.	Pareto Front Interval for Multi-objective Problems	72
5.2.3.	Number of Low Level Heuristics	74

5.2.4. Comparison of RL and RLM Heuristic Selection Methods	77
5.2.5. Comparison of Hyper-heuristic Frameworks	79
5.3. EXPERIMENTAL RESULTS FOR THE MULTI-OBJECTIVE PROBLEMS	82
5.4. EXPERIMENTAL RESULTS FOR BIN PACKING.....	94
6. CONCLUSION	98
REFERENCES	101

LIST OF FIGURES

Figure 2.1.	Single Point search based hyper-heuristic	7
Figure 2.2.	Pareto Optimal Solutions	19
Figure 3.1.	LLE LOP Graph.....	23
Figure 3.2.	Elbow Criterion	25
Figure 4.1.	Reinforcement Learning algorithm choose heuristic procedure	34
Figure 4.2.	Reinforcement Learning algorithm change score procedure	35
Figure 4.3.	Modified Reinforcement Learning algorithm change score procedure	36
Figure 4.4.	Great Deluge algorithm initialization procedure.....	37
Figure 4.5.	Great Deluge algorithm accept procedure	38
Figure 4.6.	Late Acceptance algorithm initialization procedure	39
Figure 4.7.	Late Acceptance algorithm accept procedure	39
Figure 4.8.	Swap algorithm.....	41
Figure 4.9.	Swap Low-level Heuristic	42
Figure 4.11.	Merge Low-level Heuristic	43
Figure 4.10.	Merge algorithm.....	43

Figure 4.12. Merge Tournament algorithm.....	44
Figure 4.13. Merge Tournament Low-level Heuristic	45
Figure 4.14. Finding Most Conflicting Group	46
Figure 4.15. Merge Most Conflicting algorithm	46
Figure 4.16. Merge Most Conflicting Low-level Heuristic	47
Figure 4.17. Divide algorithm	48
Figure 4.18. Divide Low-level Heuristic	49
Figure 4.19. Divide Tournament algorithm	49
Figure 4.20. Tournament algorithm	50
Figure 4.21. Divide Tournament Low-level Heuristic	51
Figure 4.22. Divide Most Conflicting algorithm	52
Figure 4.24. Change algorithm	52
Figure 4.23. Divide Most Conflicting Low-level Heuristic.....	53
Figure 4.25. Change Low-level Heuristic	54
Figure 4.26. Finding Most Suitable Group.....	55
Figure 4.27. Change from Most Conflicting to Most Suitable algorithm	55

Figure 4.28. Change From Most Conflicting To Most Suitable Low-level Heuristic	56
Figure 4.30. Change With Most Suitable Low-level Heuristic	57
Figure 4.29. Change with Most Suitable algorithm.....	57
Figure 4.31. Repair Algorithm for infeasible candidate solutions	59
Figure 4.32. Generic Hyper-heuristic Flow Chart	61
Figure 4.33. Cyclic Candidate Solution Selection Hyper-heuristic Flow Chart	63
Figure 4.34. Apply to All Candidate Solutions Hyper-heuristic Flow Chart	64
Figure 4.35. Single objective Hyper-heuristic Flow Chart	66
Figure 5.1. Graph Coloring Ranking based on the Best Solutions	88
Figure 5.2. Data Clustering Ranking based on the Best Solutions	88
Figure 5.3. Exam Timetabling Ranking based on Best Solutions	89
Figure 5.4. Pareto front of iris data with hyper-heuristic RL–LACC	93
Figure 5.5. Pareto front of synthetic2 data with hyper-heuristic SRAN–LACC	93
Figure 5.6. Bin Packing Problem Ranking based on Best Solutions	95

LIST OF TABLES

Table 2.1.	Application domains of methodologies to select perturbation heuristics.....	6
Table 2.2.	Heuristic selection and move acceptance methods.	10
Table 5.1.	The characteristics of the problem instances from the DIMACS suite. $ V $ is the number of vertices, $ E $ is the number of edges, % is the edge density and $x(G)$ is the chromatic number.	68
Table 5.2.	The characteristics of the problem instances from the Toronto benchmark suite.	70
Table 5.3.	The characteristics of the real-world data clustering problem instances.	70
Table 5.4.	The characteristics of the synthetic data clustering problem instances.	71
Table 5.5.	Tournament Size Comparison	73
Table 5.6.	Pareto Front Interval Comparison	75
Table 5.7.	Number of Heuristics Comparison.....	76
Table 5.8.	The performance comparison of RL and RLM heuristic selection methods.	78
Table 5.9.	Hyper-heuristic types average performance comparison on Exam Timetabling Problems	80
Table 5.10.	Hyper-heuristic types best performance comparison on Exam Timetabling Problems	81

Table 5.11. Hyper-heuristic types average performance comparison on Graph Coloring Problems	83
Table 5.12. Hyper-heuristic types best performance comparison on Graph Coloring Problems	84
Table 5.13. Hyper-heuristic types average performance comparison on Data Clustering Problems	85
Table 5.14. Hyper-heuristic types best performance comparison on Data Clustering Problems	86
Table 5.15. Graph Coloring best colorings comparison.....	90
Table 5.16. Exam Timetabling best colorings comparison.....	91
Table 5.17. Data Clustering best solutions comparison	92
Table 5.18. Bin Packing average best solutions comparison.....	94
Table 5.19. Pair-wise comparison of hyper-heuristics	96
Table 5.20. Bin packing problem mean number of bins comparison.....	97

LIST OF SYMBOLS/ABBREVIATIONS

ATACSH	Apply to All Candidate Solutions Hyper-heuristic
BPP	Bin Packing Problem
CCSSH	Cyclic Candidate Solution Selection Hyper-heuristic
CFMCTMS	Change from Most Conflicting to Most Suitable
ChangeMS	Change with Most Suitable
DivideMC	Divide Most Conflicting
EMC	Exponential Monte Carlo
EMCQ	Exponential Monte Carlo with counter
GCP	Graph Coloring Problem
GDEL	Great Deluge
GE	Group Encoding
GPX-CB	Greedy Partition Crossover Cardinality Based
GPX-LI	Greedy Partition Crossover Lowest Index
HGGA	Hybrid Grouping Genetic Algorithm
IEQ	Improving or Equal
LACC	Late Acceptance
LIFX	Lowest Index First Crossover
LIMX	Lowest Index Max Crossover
LLE	Linear Linkage Encoding
LLE-b	Linear Linkage Encoding With Backward Node Links
LLE-e	Linear Linkage Encoding With Ending Node Links
LMC	Linear Monte Carlo
LOP	Labeled Oriented Pseudo
MergeMC	Merge Most Conflicting
MOEA	Multi-objective Evolutionary Algorithm
MOP	Multi-objective Optimization Problem
MTP	Martello and Toth's branch-and-bound reduction algorithm
NE	Number Encoding
NSGA	Non-dominated Genetic Algorithm

RL	Reinforcement Learning
RLF	Recursive Largest Fit
RL-GDEL	Reinforcement Learning - Great Deluge Hyper-heuristic
RL-IEQ	Reinforcement Learning - Improve or Equal Hyper-heuristic
RL-LACC	Reinforcement Learning - Late Acceptance Hyper-heuristic
RLM	Modified Reinforcement Learning
RLM-GDEL	Modified Reinforcement Learning - Great Deluge Hyper-heuristic
RLM-IEQ	Modified Reinforcement Learning - Improve or Equal Hyper-heuristic
RLM-LACC	Modified Reinforcement Learning - Late Acceptance Hyper-heuristic
RLTS	Reinforcement Learning with Tabu Search
SPEA	Strength Pareto Approach
SRAN	Simple Random
SRAN-GDEL	Simple Random - Great Deluge Hyper-heuristic
SRAN-IEQ	Simple Random - Improve or Equal Hyper-heuristic
SRAN-LACC	Simple Random - Late Acceptance
VEGA	Vector Evaluated Genetic Algorithm
VNS	Variable Neighborhood Search

1. INTRODUCTION

In general, exact methods fail in solving NP-hard/complete real-world computational search problems and the researchers resort to the heuristic methods which are “rule of thumb” for solving such problems to obtain solutions with acceptable quality. The state-of-the-art heuristic methods for solving real-world problems are highly problem specific methods. They are mostly tuned for a problem in hand which is an expensive process considering both the development effort and maintenance. Moreover, there are many different heuristics which are successful in solving different problems from a given domain. Heuristics are mostly discarded and new methods have to be designed for a new problem domain. The difficulties that appear when solving problems with heuristics are mainly due to the significant range of parameter and algorithm choices which are difficult to deal with, since there is no specific and conceded way to select the best parameter values and algorithm components. These make it harder to even to choose the best heuristic approach for a given instance.

Hyper-heuristics are emerging methodologies that perform search over the space of heuristics rather than solutions [1]. A key goal of the hyper-heuristic research is to support the development of intelligent methodologies to solve problems with different characteristics from a given domain or a range of domains. This way, hyper-heuristics aim to increase the level of generality of the problem solvers. There are two main types of hyper-heuristics: methodologies to generate heuristics and methodologies to select heuristics [2]. The focus of this thesis is the latter one. A selection hyper-heuristic is a high level method which attempts to improve an initially generated candidate solution iteratively in a single point based framework by controlling (mixing) a set of low level heuristics during the search process [3]. At each iteration, a heuristic is selected from the set of low level heuristics first and then it is applied to the candidate solution in hand, generating a new one. Then, a decision is made whether to accept or reject the new solution. The selection hyper-heuristics aim to combine the strengths of different heuristics while avoiding from their weaknesses for a given problem to raise the level of generality.

The grouping problems are concerned with partitioning a set of items into a collection

of disjoint subsets [4]. All grouping problems have their own distinguishing set of constraints. Some of the grouping problems can be formulated as a multi-objective problem where objectives creates a trade-off or as a single objective problem. The examples of multi-objective grouping problems are graph coloring, data clustering and exam timetabling. The examples of single-objective grouping problems are bin packing and stock cutting. The representation of candidate solutions in grouping problems is a critical issue, because some representation schemes create symmetries in the search space and these symmetries cause different solutions to represent the same point in the search space, so the search space increases undesirably.

In this thesis, a general grouping selection hyper-heuristic framework based on linear linkage encoding is described. The performance of a range of selection hyper-heuristics is investigated over a set of problem instances including Toronto Benchmark [5] and DIMACS Challenge Suite [6] for exam timetabling and graph coloring. Several real-world and synthetic data sets are used for data clustering and data sets provided by Falkenauer [7] are used for bin packing.

This thesis is organized as follows: Chapter 2 summarizes the hyper-heuristic and multi-objective optimization concepts. Chapter 3 presents an overview of grouping problems, particularly graph coloring, exam timetabling, data clustering and bin packing along with linear linkage representation. Chapter 4 describes the grouping hyper-heuristic framework, including the heuristic selection and move acceptance methods, low-level heuristics, objective functions used for each problem domain. Chapter 5 discusses the experimental results obtained for each problem domain using the proposed hyper-heuristics. Finally, conclusions are provided in chapter 6.

2. BACKGROUND

2.1. HYPER-HEURISTICS

2.1.1. The Hyper-heuristic Concept

Automating the design and tuning heuristic methods to solve hard computational search problems is one of the key motivations for hyper-heuristics([8–10]). Heuristics and meta-heuristic methods are problem dependent, on the other hand hyper-heuristics aim to develop more generally applicable problem independent methods. Hyper-heuristics aim to find the right method or sequence of heuristics in a given situation rather than trying to solve the problem directly. One of the important goals of hyper-heuristics is to devise easy-to-implement low-level heuristics to design generic methods for all problems to find acceptable optimal solutions. A hyper-heuristic can be seen as a high-level methodology that is built on top of the low-level heuristics to solve the given problem. It automatically produces necessary combination of its components for the given problem instance. Two main ideas provide inspiration for different types of hyper-heuristic frameworks [2]:

- Selecting and designing efficient hybrid and/or cooperative heuristics are computational search problems in itself.
- Learning mechanisms can improve the search methodologies.

[11] first used the term hyper-heuristic and then [12] used hyper-heuristic term for combinatorial optimization and describe it as “heuristic to choose heuristics” [13] is the first journal that hyper-heuristic term is used.

In 2003, the first review book chapter on hyper-heuristics, stressed one of the key objectives of the hyper-heuristics as to raise the level of generality [8]. [10] published a tutorial article which gives useful implementation approaches for hyper-heuristics and specifies some research issues and new application domains to apply hyper-heuristics. Chakhlevitch and Cowling in [14] interested in recent developments in hyper-

heuristics rather than the historical improvement of hyper-heuristics and heuristic generation methodologies. They classify and discuss the hyper-heuristics recently developed. Generating new heuristics from a set of potential heuristic components is discussed in [9]. In this approach Genetic Programming is an important part of the discussion. This chapter gives detailed description of the approach with some case studies. It also includes a discussion of known issues for this type of hyper-heuristic. [9] represents a unified classification and definition of hyper-heuristics based on previous categorizations. Based on this categorization, there are two main hyper-heuristic categories: heuristic *selection* and heuristic *generation*.

2.1.2. Classification of Hyper-heuristics

In [1], hyper-heuristic is defined as follows:

A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems

In [9], classification of hyper-heuristics is proposed according to two dimensions:

- the nature of the heuristic search space
- the source of feedback during learning

Different heuristic search spaces can be combined with different sources of feedback and different learning techniques.

There are two main categories of hyper-heuristics based on the nature of the search space, these are:

- Heuristic selection: methodologies for choosing or selecting existing heuristics
- Heuristic generation: methodologies for generating new heuristics from components of existing ones

Heuristics can be categorized as construction heuristics and perturbation heuristics

under that two main categories. This categorization is about the low-level heuristics used in the hyper-heuristic frameworks.

Hyper-heuristics can be also classified with their learning mechanisms as learning hyper-heuristics and non-learning hyper-heuristics. The distinction between these two is using some feedback mechanism from the search process or not. Therefore learning hyper-heuristics uses some feedback from the search process, on the other hand non-learning hyper-heuristics do not use any feedback from the search process. Learning hyper-heuristics are also divided into two categories which are *online* learning hyper-heuristics and *offline* learning hyper-heuristics.

- Online learning hyper-heuristics: The learning process takes place when the algorithm is on the run. Therefore all the problem instance dependent local properties can be used in the learning process. Hyper-heuristics use these local properties to select appropriate low-level heuristic. Reinforcement Learning is one of the examples of the online learning hyper-heuristics. In this research, reinforcement learning is used and detailed information of this algorithm can be found in Section 4.2.2
- Offline learning hyper-heuristics: The idea is to train the algorithm with a set of training instances before running the algorithm for the real problem. Examples of offline learning hyper-heuristics are: learning classifier systems, case-base reasoning and genetic programming.

To sum up all the classifications:

- Nature of heuristic search space
 - **Heuristic selection methodologies**
 - * Construction heuristics
 - * Perturbation heuristics
 - **Heuristic generation methodologies**
 - * Construction heuristics
 - * Perturbation heuristics
- Feedback during learning

Table 2.1. Application domains of methodologies to select perturbation heuristics.

Application domain	Reference(s)
Channel assignment	[15, 16]
Component placement	[17]
Personnel scheduling	[12, 13, 18–20]
Packing	[20, 21]
Planning	[22]
Reactive Power Compensation	[23]
Space allocation	[24–26]
Timetabling	[13, 20, 27–29]
Vehicle routing problems	[30]

- **Online learning hyper-heuristics**
- **Offline learning hyper-heuristics**
- **Hyper-heuristics without learning**

In this study, we focus on heuristic selection methodologies managing a set of perturbation low-level heuristics. The low-level heuristics for grouping problems are described in Section 4.4 along with the online learning hyper-heuristic based on reinforcement learning (Section 4.2.2).

2.1.3. Selection Hyper-heuristics: Heuristic Selection and Move Acceptance Methods

In selection hyper-heuristic frameworks, a heuristic (or a subset of heuristics) is automatically selected and applied to a candidate solution to improve it. Heuristic selection mechanisms devise some offline and online learning mechanism to make better decisions. These approaches are applied to a wide variety of combinatorial optimization problems, some of them are listed in Table 2.1 .

The execution process of single point search based hyper-heuristic framework is shown in Figure 2.1. In this framework, search starts with an initial candidate solution and this candidate solution ($s_{current}$) is improved until some termination criteria are fulfilled. For every step the procedure is as follows; firstly, a heuristic (h) is selected from a set of low-level heuristics and it is applied to the current candidate solution. Then a decision is made whether to accept or reject the new candidate solution (s_{new}). If the new candidate solution is accepted, it is the new current candidate solution, otherwise candidate solution stays the same for the next run. The single point based search hyper-heuristics are composed of two key components:

- heuristic selection method
- move acceptance method

These are identified in [28] and [3]. New hyper-heuristics can be constructed by using different combinations of heuristic selection methods and move acceptance methods. When a heuristic selection method is changed with another one, we have a new hyper-heuristic.

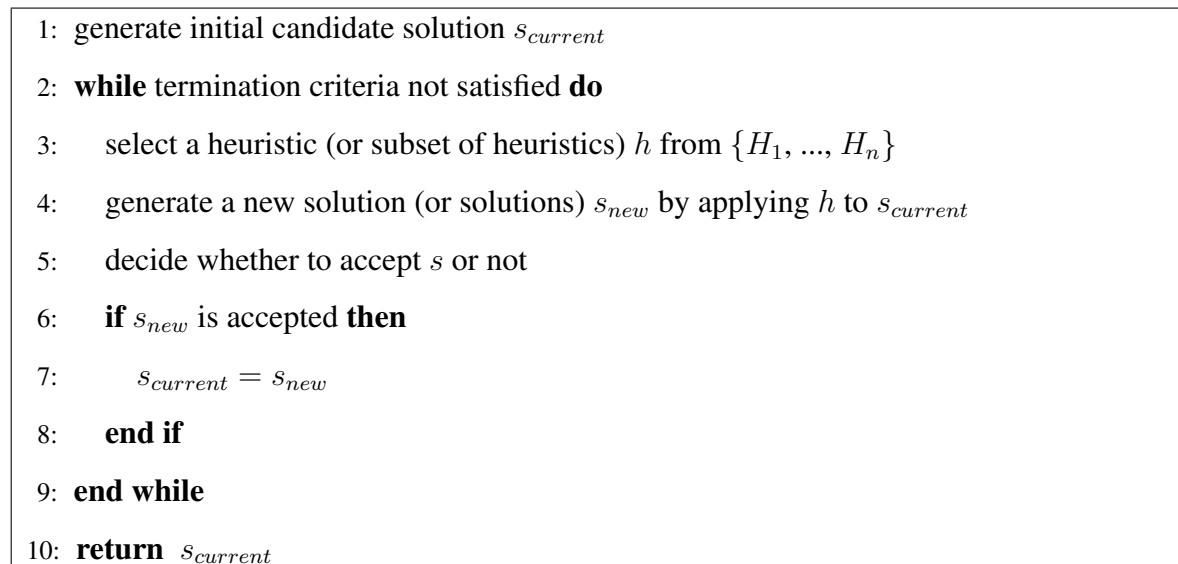


Figure 2.1. Single Point search based hyper-heuristic

Learning mechanisms, that heuristic selection mechanisms use, are divided into three

parts as stated before, these are online learning mechanisms, offline learning mechanism and non-learning mechanisms. Non-learning heuristic selection methods are based on a random or an exhaustive process instead of a learning process. On the other hand, the learning mechanisms aim is to improve the decision making process in the heuristic selection. Most of the hyper-heuristics use online learning rather than offline learning mechanisms. Online learning mechanisms use online scores that are generated based on the performances of the low-level heuristics. These scores are used in a systematic way at each step to select the appropriate low-level heuristic to be applied on the candidate solution. Score based online learning heuristic selection mechanisms require five main components. These are:

- initial scoring
- memory length adjustment
- strategy for heuristic selection based on the scores
- score update rule for improvement
- score update rule for worsening

Initial scoring may affect the performance of a hyper-heuristic, generally all low-level heuristics are assigned with the same initial score. The initial score might be 0 or some other value based on the implementation of the hyper-heuristic. Memory length determines the affect of the previous performance of a heuristic. Higher scores increase the probability of selection of the low-level heuristic, in some strategies the low-level heuristic with the highest score is selected.

Rewarding and punishing the low-level heuristics based on their individual performance during the problem solution is called score mechanism. If a low-level heuristic improves the candidate solution, it is rewarded by the predefined update rule. On the other hand, low-level heuristics that worsens the current candidate solution is punished by the predefined update rule. Reward increases the score of the low-level heuristic, however punishment decreases the score of the low-level heuristic.

The acceptance strategy is another important component in hyper-heuristics. Acceptance strategies can be categorized as *deterministic* and *non-deterministic*.

Deterministic acceptance strategies make the same decision if the candidate solution and the state of the problem is the same. Non-deterministic acceptance strategies may give different decisions with the same candidate solution and the state. Non-deterministic methods require different additional parameters for decision making, time is one of the most common parameter used in non-deterministic acceptance methods. Table 2.2 summarizes the most commonly used heuristic selection and move acceptance methods.

In this research, Simple Random heuristic selection method is used as a heuristic selection with no learning method and Reinforcement Learning heuristics selection method is used as a heuristic selection with learning. Also Improve and Equal deterministic acceptance method and Great Deluge and Late Acceptance non-deterministic acceptance methods are used as the acceptance methods.

Reinforcement learning is one of the most commonly used method in hyper-heuristics, for more detailed information about *reinforcement learning* see [34] and [35]. A reinforcement learning strategy is also used in this study, for the implementation details you can see Section 4.2.2.

Reinforcement Learning is used as a heuristic selection mechanism in [22]. Each low-level heuristic's score is updated according to a predetermined update rule which is based on heuristic's performance at each decision point. The heuristic selection mechanism selects the appropriate heuristic based on these scores. All Moves acceptance method is used in the study, therefore when a heuristic is chosen to be applied, it is accepted. Different score update mechanism are investigated in this work. This study suggests that combining a low rate of adaptation for rewarding an improving move and strong rate of adaptation for punishing a worsening move is a good choice. In addition to this using max strategy when choosing a heuristic at each step often generates better result when compared to choosing a heuristic with a roulette wheel scheme using the scores.

Similar to Nareyek's study, [13] is based on the score mechanisms by learning to select among low-level heuristics. They presented a *Reinforcement Learning* with *Tabu Search*. A dynamic tabu list of low-level heuristics also included in the hyper-heuristic, this list

Table 2.2. Heuristic selection and move acceptance methods.

Component Name	Reference(s)
Heuristic selection	
Simple Random	[12, 31]
Random Permutation	[12, 31]
Greedy	[12, 18, 31]
Peckish	[18]
Random Gradient	[12, 31]
Random Permutation Gradient	[12, 31]
Choice Function	[12, 31]
Reinforcement Learning	[22]
Reinforcement Learning with Tabu Search	[13, 21]
Move acceptance	
All Moves	[12, 31]
Only Improvements	[12, 31]
Improving and Equal	[12, 31]
Monte Carlo	[17]
Great Deluge	[15, 28]
Record to Record Travel	[16]
Tabu Search	[32]
Simulated Annealing	[25, 28]
Simulated Annealing with Reheating	[20, 21]
Late Acceptance	[33]

temporarily excludes them from the available heuristics. The algorithm deterministically selects the highest ranked low-level heuristic that is not in the tabu list. All Moves acceptance mechanism is used in this study, therefore when the heuristic is chosen the final candidate solution is accepted whether it improves the solution or not. The improvement increases the heuristic rank, on the other hand if there is no improvement both the heuristic rank is decreased and the heuristic is enqueued into the tabu list. This decreases the probability of the non-improving heuristics. When a non-improving move is accepted the tabu list is emptied. University course timetabling and nurse rostering problems are used to evaluate this hyper-heuristic. The results are competitive with the state-of-art problem specific techniques. [24] extends this methodology to be used in multi-objective optimization.

A sales summit and a project presentation problem which are real-world scheduling problems are used in [12, 31] to propose and compare a variety of the hyper-heuristic components. In [12], a score based heuristic selection method *Choice Function* based on reinforcement learning is introduced. The function adaptively ranks the low-level heuristics based on their score. *Simple Random* selects a heuristic from low-level heuristics completely randomly at each step. Simple Random approach is used in this study, you can find a detailed description in Section 4.2.1. *Random Gradient* is a variant of Simple Random, it randomly chooses a heuristic among low-level heuristics and applies this heuristic to the candidate solution until it doesn't improve the solution. *Random Permutation* generates a random ordering of the low-level heuristics and the heuristics are applied according to this randomly defined order, at each step the heuristic which is the next one in the order is applied. *Random Permutation Gradient* is a variant of the Random Permutation, the relation of Random Permutation and Random Permutation Gradient is same with the Simple Random and Random Gradient. Low-level heuristics are in a randomly generated order, however the next heuristic is applied to the solution when the heuristic could not improve the solution. *Greedy* applies all low-level heuristics on the current candidate solution and it selects the best improved solution. Random Gradient and Random Permutation Gradient can be considered as intelligent heuristic selection mechanisms. Both of them implement some kind of Reinforcement Learning by selecting the improving heuristic in the next step until it doesn't improve. This type of strategy can be useful if there are not many plateaus in the search landscape. These heuristic selection mechanisms execute fast, Greedy which runs in

an exhaustive manner, is the only exception for this case.

All Moves and *Only Improvements* were considered as the deterministic acceptance criteria. Choice Function—All Moves hyper-heuristic resulted promising in the experimental results in [12]. A series of experiments run and manual parameter tuning is done to obtain the best parameter set. In [31] a variant of Choice function is proposed, which uses reinforcement learning to automatically update all parameters at each step. This new variant of Choice function outperforms the simple Choice Function with manually defined parameters.

Simple Random and Greedy heuristic selection methods are used in [18]. There is a probability that all moves may worsen the solution but Greedy method described in [12] does not accept any worsening moves. This study accepts worsening moves in this situation. Peckish heuristic selection strategies are also studied with four different Tabu Search based move acceptance methods. Only Improving, All Moves and a variant of All Moves are used as move acceptance criteria in the study. The All Moves variant discards solutions that are same with the current solution. Real-world personnel scheduling problem is used to evaluate the hyper-heuristic. Ninety five low-level heuristics are used and the results are promising. This large set of low-level heuristics reveal a problem that selecting a low-level heuristic becomes slower. This problem inspired a new study in [32] to represent two new learning strategies for choosing the subset of the fittest low-level heuristics. The results of this study reveal that Greedy-Tabu Search is the most promising hyper-heuristic among them. Greedy-Tabu Search linearly reduces the number of the fittest low-level heuristics.

[17] proposed three different types of Monte Carlo acceptance strategy; *Linear* Monte Carlo(LMC), *Exponential* Monte Carlo (EMC) which is based on an exponential probability function and *Exponential* Monte Carlo with *counter* (EMCQ) which is based on the computation time and a counter of consecutive non-improvement iterations. Optimization of the scheduling of electronic component placement on a printed circuit board is used to evaluate the hyper-heuristics that are combination of Simple Random with one of the Monte Carlo acceptance strategies (LMC, EMC, EMCQ). The performance of these hyper-heuristics are compared with combination of (Simple Random, Choice Function) and (All

Moves, Only Improving) hyper-heuristics. Simple Random–EMCQ hyper-heuristic has a superior performance over the deterministic acceptance with and without learning hyper-heuristics in the problem instances tested in this study. Different results may occur for the same number of iterations because faster machines executes more instructions in a given time.

Great Deluge acceptance criteria accepts moves at each decision point if it is not worse than the expected objective value which changes at linear rate every step. If $f(s_{new}) < level = f(s_0) - (t\Delta F)/T$ then the move is accepted at step t , where $f(s_0)$ is the objective value of the initial solution. ΔF is the difference between the objective values of $f(s_0)$ and the expected final objective value and T is the maximum number of iterations. A variant of *Great Deluge* acceptance criteria used in [15] and Simple Random heuristic selection strategy is used in this study. A real-world channel assignment benchmark problems are used to evaluate this hyper-heuristics. Hyper-heuristic is compared with a constructive heuristic and a genetic algorithm, the results are competitive.

[36] presented an acceptance criteria named *Record-to-Record Travel* and [16] modified this acceptance criteria for using in hyper-heuristics. Channel assignment problem is used to evaluate the hyper-heuristic that uses Simple Random for the heuristic selection and it is compared with All Moves, Only Improving and EMCQ move acceptance strategies. The proposed method has better results. It is also compared with the state-of-the-art algorithm for this problem; a constructive heuristic and a genetic algorithm, Simple Random–Record-to-Record Travel heuristic results comparable with these previous studies.

[25] introduced Simulated Annealing in hyper-heuristics and they discuss the automation of the Simulated Annealing parameters. Simple Random–Simulated Annealing hyper-heuristics outperformed the following according to the results; Simple Random–Only Improving, Simple Random–All Moves, Greedy–Only Improving, Choice Function–All Moves and two conventional simulated annealing approaches.

A multi-objective Simple Random–Simulated Annealing hyper-heuristic is described in [23]. Power compensation problem in electricity distribution networks is the problem used

with the proposed hyper-heuristic. Six low-level heuristics are designed to make a move from one feasible solution to another. Simulated Annealing makes the decision based on the dominance between these two feasible solutions. The weighted sum of two objective values is used in acceptance probability. The proposed Simple Random–Simulated Annealing hyper-heuristic performed slightly worse than a multi-objective genetic algorithm.

A variant of Reinforcement Learning with Tabu search (RLTS) is hybridized with a *Simulated Annealing with Reheating* move acceptance strategy in [21]. Score updates and acceptance rate reductions perform together which are the results of RLTS and Simulated Annealing with Reheating respectively. A packing problem with real-world data is used to evaluate the proposed hyper-heuristic. The hyper-heuristic which utilize Reinforcement Learning with Tabu Search as heuristic selection mechanism and Simulated Annealing with Reheating as move acceptance method is superior in performance than a simpler local search strategy.

[20] uses a reinforcement learning mechanism with a short term memory as a heuristic selection mechanism in his proposed hyper-heuristic. Low-level heuristics scores are used to calculate the selection probabilities and they are selected with a roulette wheel strategy after these probabilities are calculated. As a move acceptance method, a variant of Simulated Annealing with Reheating which executes switching between annealing and reheating phases during the search, is used. Nurse rostering, course timetabling and bin packing problems are used to evaluate the proposed hyper-heuristic and its results are competitive compared to previous studies.

Late Acceptance (LACC) strategy is a memory based technique, it devises a list of size L to hold the history of objective values from the previous solutions. New candidate solution's objective value is compared with an objective value from the history and based on this comparison it is decided to accept or reject the new candidate solution. [33] used this acceptance criteria to investigate the performance of hyper-heuristics combined with different heuristic selection methods. Simple Random, Greedy, Choice Function, Reinforcement Learning and Reinforcement Learning with Tabu Search heuristic selection methods are used. Simple Random performed the best with Late Acceptance and Greedy

performed the worst with Late Acceptance. The delay in comparison of objective values seem to be deceiving in the learning mechanism.

2.1.4. Performance Comparison Studies

In [28] previously proposed heuristic selection and move acceptance mechanisms are combined in 35 different hyper-heuristics and they are tested for comparison. Heuristic selection mechanisms used in this research are namely: Simple Random, Random Gradient, Random Permutation, Random Permutation Gradient, Greedy, Choice Function and Reinforcement Learning with Tabu Search. Move acceptance mechanisms used in this research are namely: All Moves, Only Improving, Improving and Equal, Great Deluge and EMCQ. Hyper-heuristics that use the Improve or Equal move acceptance strategy performed slightly better than the other hyper-heuristics over a set of benchmark functions. When heuristic selection mechanisms are compared, the Choice Function produced a slightly better average performance. The results reveal that move acceptance strategies play more important role compared to heuristic selection mechanisms. Over the exam timetabling instances, Choice Function–EMCQ and Simple Random–Great Deluge outperformed the rest of the hyper-heuristics.

[37] proposed four different types of hyper-heuristics and investigate the performance of these frameworks. First hyper-heuristic type is the standard hyper-heuristic called F_A in the research. The low-level heuristics, which are composed of *hill climber* heuristics which aim to improve the solution at each step and *mutational* heuristics which perturb a candidate solution without considering whether the new solution will be improved or not, are treated equally in one set. Hyper-heuristic selects one of them based on its heuristic selection methodology. The second hyper-heuristic which is called F_B in the study is a variant of F_A . If the selected heuristic is a hill climber heuristic, move acceptance methodology applies as in F_A , but if a mutational heuristic is selected, a predefined hill climber heuristic is applied after the mutational heuristic is applied to the candidate solution. Third hyper-heuristic framework that is called F_C in this study uses only mutational heuristics as its low-level heuristic set. One of the mutational low-level heuristic is selected and applied to the candidate solution and then a predefined hill climber heuristic is applied to the candidate solution. The last

one that is called F_D hyper-heuristic framework uses two hyper-heuristics successively for managing mutational and hill climber heuristics. Test results reveal that F_C outperforms the other three hyper-heuristic frameworks.

[3] extend the studies [28] and [37]. When learning mechanisms used, it is observed that some low-level heuristics are rarely chosen. The repeated experiments show that the choice of single hill climber heuristic that is used in every iteration affects the performance of the F_B and F_C frameworks. When the best hill climber heuristic is chosen, F_B and F_C outperformed others, F_C also outperforms genetic algorithm and its performance is comparable with memetic algorithms.

[26] worked on a fresh produce inventory and shelf space allocation problem and compared the results against a multi-start generalized reduced gradient algorithm proposed in [38], some hyper-heuristics and meta-heuristics. The empirical results show that Simulated Annealing based hyper-heuristics performs better than Reinforcement Learning with Tabu Search—All Moves hyper-heuristic and similar performances to the traditional Simulated Annealing and GRASP. Reinforcement Learning with Tabu Search outperforms Simple Random when the heuristic selection methods are compared in hyper-heuristic frameworks.

[20] treats the hyper-heuristic components, heuristic selection and move acceptance as indivisible and tries to update their parameters together. [39] showed that they are separable in the study which combines the heuristic selection methods, Simple Random and Choice Function with move acceptance methods, Simulated Annealing, Simulated Annealing with Reheating and EMCQ. These combined hyper-heuristics are tested with Greedy—Simulated Annealing with Reheating and the hyper-heuristic proposed in [20]. These are tested on exam timetabling problems and the results show that Choice Function—Simulated Annealing outperforms the others. [22] shows that using max strategy for heuristic selection performs better. This explains why Choice Function performed better than the learning mechanism in [20]. This mechanism rewards the heuristic when it is accepted, it doesn't consider the improvement in the candidate solution after the low-level heuristic is applied.

2.1.5. Hyper-heuristic Tools

Recently, hyper-heuristic software libraries have been implemented for rapid development and research: Hyperion [40] and Hyflex [41]. Hyperion provides a general recursive framework for the development of hyper-heuristics (or meta-heuristics), supporting the selection hyper-heuristic frameworks provided in [3]. Hyflex provides reusable hyper-heuristic (meta-heuristic) components, having a support for the problem domains of Boolean Satisfiability (MAX-SAT), One Dimensional Bin Packing, Permutation Flow Shop (PFS) and Personnel Scheduling (PS) each with ten different instances and a set of low-level heuristics. Burke et al. [42] investigated the performance of a range of selection hyper-heuristics implemented as part of HyFlex. This was a proof of concept study for CHeSC: Cross-Domain Heuristic Search Competition ¹. The best selection hyper-heuristic will be determined among CHeSC competitors which generalizes the best across a set of problem instances from different problem domains. Burke et al. [42] reported that the best performing hyper-heuristic was an iterated local search approach. This result also shows that the F_C framework has a lot of potential. More on hyper-heuristics can be found in [1, 2, 8–10, 14]

2.2. MULTI-OBJECTIVE OPTIMIZATION

Optimization can be described as minimization or maximization of a real function. An allowed set of real and/or integer variables are chosen systematically to maximize or minimize the real function. Optimization can be categorized as single-objective and multi-objective. Multi-objective optimization involves more than one objective to minimize or maximize, generally these objectives conflict with each other.

The Multi-Objective Optimization Problem (MOP) can be defined as the problem of finding a vector of decision variables which satisfies constraints and optimizes them as a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other [43]. The term “*optimize*” means finding a solution which would give the values of all objective functions acceptable to the decision maker [44]

¹<http://www.asap.cs.nott.ac.uk/chesc2011/>

The notion of “*optimum*” that is mostly adopted in the literature is firstly proposed by Francis Ysidro Edgeworth in 1881. In 1896 Vilfredo Pareto generalized this notion. Therefore this notion firstly called in literature as Edgeworth-Pareto optimum, however in later studies the notion is accepted and used as *Pareto Optimum*.

Single-objective optimization and multi-objective optimization are different in the nature of the problems that they devise to solve. Single-objective optimization aim to find the optimal solution, therefore there is only one optimal value in this optimization. On the other hand in multi-objective optimization, there are no one single optimal value. The nature of multi-objective optimization contains multi conflicting objects, therefore it results with a set of optimal values. Each different objective may have different individual optimal solutions. The term *conflicting* can be described as if there is a sufficient difference in the optimal solutions for two different objectives, then these two objectives are known as *conflicting*. The reason of having not one optimal solution but a set of optimal solutions is the conflicting objectives and the solutions can not be considered better than any other optimal solution with respect to all objective functions. This set of optimal solutions are called Pareto-Optimal solutions. This notion is illustrated with an example in Figure 2.2

In this figure, f_1 and f_2 represent two objectives to be minimized. Solution A represents a solution which is minimized with respect to objective f_1 but it has the maximum value with respect to objective f_2 . In contrary the solution D represent the minimum value for objective f_2 , but it has the maximum value for objective f_1 . If the objectives f_1 and f_2 are equally important goals, it can not be said neither solution A is better solution nor solution D . Solution A is better than solution D with respect to objective f_1 , on the other hand solution D is better than solution A with respect to objective f_2 . There exists a trade-off between these acceptable solutions.

Pareto-Optimal Set can be defined as; the non-dominated solutions P' , those are not dominated by any member of a set of solutions P . The non-dominated set of entire feasible search space S is the *global pareto optimal set*. If for every member x in a set P' there exists no solution y in the neighborhood of x dominating any member of P' , then P' is a *locally pareto-optimal set*.

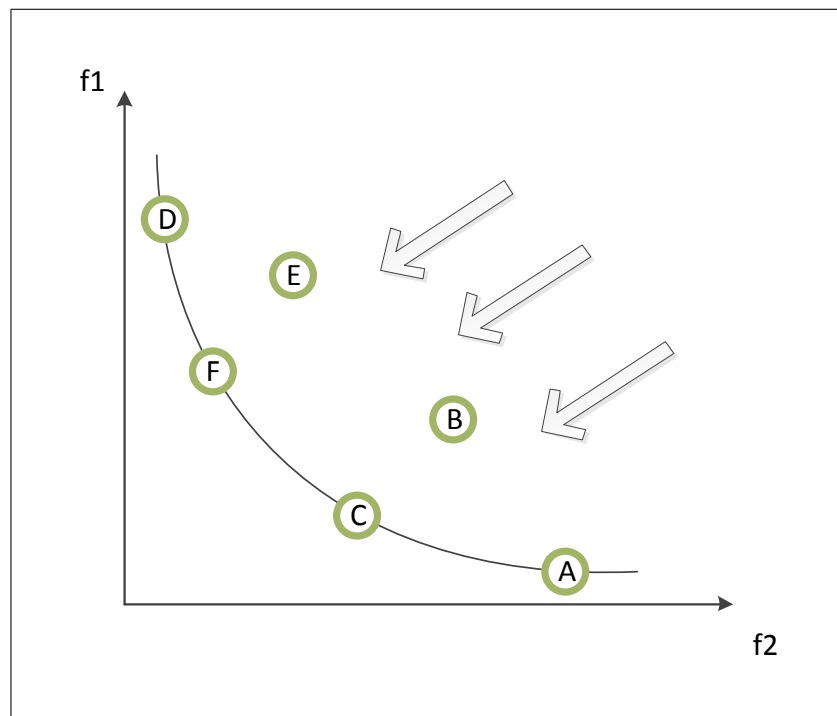


Figure 2.2. Pareto Optimal Solutions

There are two condition for a solution x^1 to dominate another solution x^2 . These conditions are:

1. The solution x^1 is no worse than x^2 in all objectives
2. The solution x^1 is strictly better than x^2 in at least one objective

If x^1 dominates x^2 , then

- x^2 is dominated by x^1
- x^1 is non-dominated by x^2
- x^1 is non-inferior to x^2

There are many classical and evolutionary methods for solving MOP. Some of them can be listed as:

- Classical Methods:

- Weighted Sum Method
- The ϵ -Perturbation Method
- Evolutionary Methods
 - Schaffer's Vector Evaluated Genetic Algorithm (VEGA) [45]
 - Fonseca and Fleming's Multi-objective Genetic Algorithm [46]
 - Horn, Nafploitis and Goldberg's Niche Pareto Genetic Algorithm [47]
 - Zitzler and Theile's Strength Pareto Approach (SPEA) [48]
 - Srinivas and Deb's Non-dominated Genetic Algorithm (NSGA) [49]

More information about multi-objective optimization can be found in [50–52].

3. GROUPING PROBLEMS

The performance of a range of selection grouping hyper-heuristics for solving grouping problems, in particular graph coloring, examination timetabling, data clustering and bin packing is investigated in this study. This chapter provides an overview of the grouping problem concepts and these problem domains.

Grouping problems can be described as the problem of searching for the best partition of a set of items into a collection of mutually disjoint subsets [4] subject to constraints.

$$V = V_1 \cup V_2 \cup V_3 \cup \dots \cup V_N \text{ and } V_i \cap V_j = \emptyset \text{ where } i \neq j \quad (3.1)$$

In a grouping problem, the aim is to partition the members of a set V into N different groups, where $(1 \leq N \leq |V|)$ and each item is in exactly one group (Equation 3.1). There are usually a set of constraints for a valid solution to a grouping problem, because in general, not all groupings are permitted. A valid (feasible) solution has to comply with these constraints. A grouping problem has its own specific constraints, for example in graph coloring, adjacent nodes can not be in the same group. In the bin packing problem, the constraints are different then the graph coloring problem, the sizes of items in a bin can not exceed the bin's capacity. Exam Timetabling also has different constraint and its constraints are also different in different problem instances, one example of exam timetabling constraint can be like, any two exam can not be at the same time if these two exams will be taken by any of the students. Exam Timetabling constraint can be divided into two categories as hard constraint and soft constraints. Hard constraints must be satisfied, while soft constraints are not mandatory but the aim is to resolve soft constraints as many as possible. Almost all grouping problems have these kind of constraints. The objective of grouping is to optimize a fitness function defined over a set of groupings. In all three examples that are given above is to minimize the number of groups while satisfying the constraints defined for each problem.

Representation of grouping problems is a challenging issue. The reason of this is the symmetries in the search spaces, these symmetries causes different solutions to be equal in the search space. This is an undesirable case, because it increases the search space which decreases the performance of the search procedures by examining the same solutions more than once. The previously used representations do not solve these symmetries in the search space issue, therefore a new representation schema name Linear Linkage Encoding (LLE) [53] is proposed to resolve this issue.

There are two commonly used representation schemes prior to LLE, these are Number Encoding (NE) and Group Encoding(GE). In NE, each object is represented with a group id which it belongs to. For example, 34112314 encodes a solution where first object is in group 3, the second object is in group 4, third object is in group 1 and so on. However the encoding 12334132 represents exactly the same solution. The naming or the ordering of the partition sets is irrelevant to the solution encoding. [4] represents the drawbacks of this representation schema and showed that it is against the minimal redundancy principal for encoding scheme [54]. On the other hand GE places the objects, which are in the same group, in the same partition set. For example, the encoding represented for NE can be encoded in GE as $(1, 6)(2, 8)(3, 4, 7)(5)$. Unlike NE, search operators work on groups rather than objects, so ordering within each partition is unimportant. However the following represents the same solution, $(3, 4, 7)(5)(1, 6)(2, 8)$ because the ordering redundancy among groups problem is not solved in this representation scheme too.

LLE is proposed to solve these representation issues. It is explained in detail in section 3.1. In this study, LLE is used as the representation scheme for solving grouping problems.

3.1. LINEAR LINKAGE ENCODING

Linear Linkage Encoding (LLE) is proposed in [53] to resolve the symmetries issue that causes redundancy in the search space. LLE can be represented using an array, each item in the array has a value between 1 to n that is a link to another object in the same partition. Two objects are in the same group if either one can be reached from another through the links. If an object links to itself, that is considered as an ending node. LLE constructs a

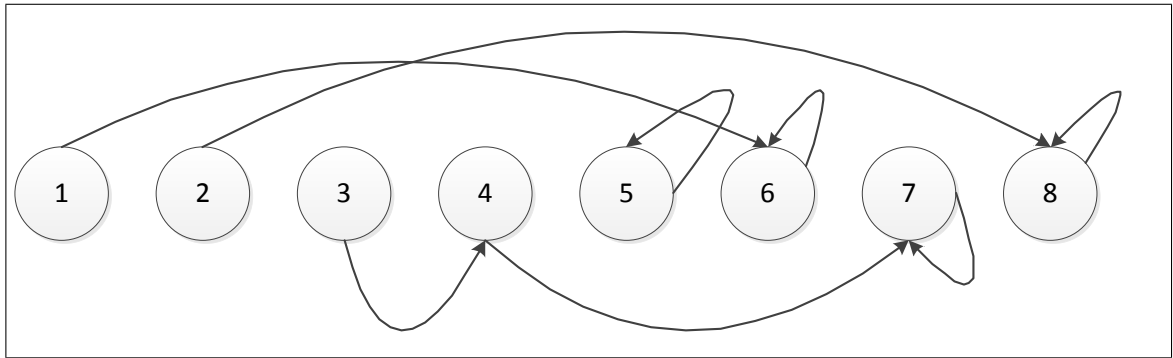


Figure 3.1. LLE LOP Graph

pseudo linear path with the only loop allowed being a self loop link to mark the last node in a group. LLE can be represented as a labeled oriented pseudo (LOP) graph which is a directed graph $G(V, E)$, where V is the vertex set and E is the edge set. An example of an LLE representation can be seen as an array representation and LOP representation in the figure 3.1

There are two requirements for LLE to be feasible. First, backward links are not allowed, this means that all nodes in a group should point to a node with a higher index than itself. Second requirement is that no two nodes can point to the same node in the array, only exception to this case is the ending node which points to itself. A composition of G is a grouping of $V(G)$ into disjointed oriented pseudo path graphs G_1, G_2, \dots, G_m with the following properties:

- Disjoint paths $\cup_{i=1}^m V(G_i) = V(G)$ and for $i \neq j, V(G_i) \cap V(G_j) = \emptyset$
- Non-backward oriented edges: If there is an edge e directed from vertex v_i to v_k then $i \leq k$
- Balanced Connectivity
 1. $|E(G)| = |V(G)|$
 2. each G_i has only one ending node with an *in-degree* of 2 and *out-degree* of 1
 3. each G_i has only one starting node whose *in-degree* = 0 and *out-degree* = 1
- All other $|V(G_i)| - 2$ vertices in G_i have *in-degree* = *out-degree* = 1

There are several studies that use LLE for solving grouping problems. [55] used LLE to solve two-level clustering problem with a genetic algorithm and used large datasets. The results were promising and competitive with the supervised techniques. [56] worked on two grouping problem domains, Graph Coloring and Timetabling. This study used a multi-objective genetic algorithm to solve these problems and investigate the performance of LLE on these problem domains. The genetic algorithm operators Lowest Index First Crossover (LIFX) and Lowest Index Max Crossover (LIMX) have promising result. [33] worked on Examination Timetabling problem with a newly proposed acceptance criteria Late Acceptance (LACC) [57]. Perturbative hyper-heuristic framework used in this study and best heuristic selection match is investigated for LACC. Simple Random heuristic selection performed well with LACC according to the experiments. Another observation of the study is that learning mechanisms based on reinforcement learning or statistical analyses do not function well with LACC. [58] studied on the representation issues in Graph Coloring and it proposed two new versions of LLE which are Linear Linkage Encoding with Ending Node Links and Linear Linkage Encoding with Backward Links. [59] used LLE for solving bin packing problems with grouping genetic algorithm and the performance of custom made LLE operators are compared with traditional recombination operators.

3.2. ELBOW CRITERION

In this study, data clustering, exam timetabling and graph colouring are formulated as multi-objective problems. The minimum number of clusters causing the least error, the minimum number of timeslots causing the least violations and the minimum number of colours causing the least conflicts are searched. All these objectives are conflicting objectives and finding the minimum number of groups (clusters, timeslots, colours) is a distinct issue from the process of actually solving the grouping problem. The elbow criterion is a common rule of thumb to determine the best number of groups against a given conflicting objective. The elbow criterion claims that a number of groups should be chosen where adding another group does not create a better solution. In Figure 3.2, it is clearly shown that adding a group to the solutions of 2 groups and 3 groups have a significant effect in fitness value assuming a minimization problem. However, when number of groups is increased from 4 to 5, it has a very slight difference compared to its successors. Therefore, the solution with 4 number of

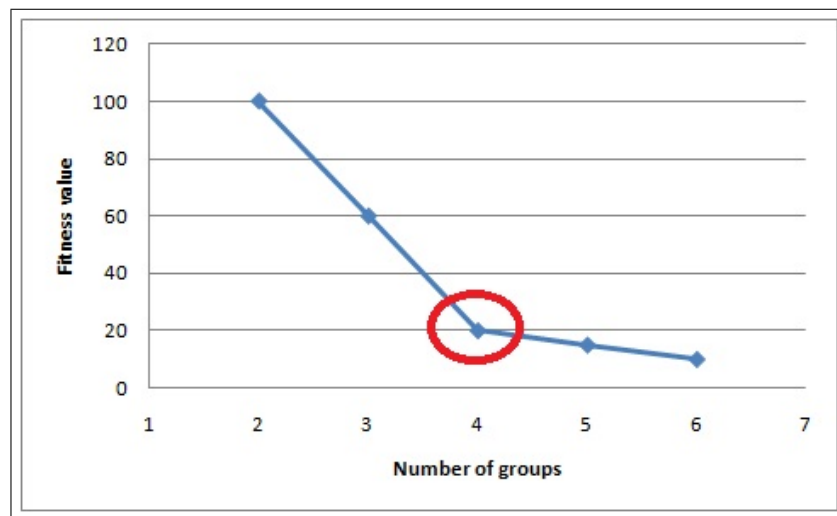


Figure 3.2. Elbow Criterion

groups should be selected in this example according to the elbow criterion.

In this study, elbow criterion is used to determine the best solution after running the grouping hyper-heuristic on a problem instance. The result of multi-objective grouping hyper-heuristics constitutes a pareto front for non-dominating list of candidate solutions and the best solution from this list is selected based on the elbow criterion.

3.3. DATA CLUSTERING

Data Clustering is a kind of a grouping problem where patterns like observations, data items or feature vectors are classified into groups which are called clusters in the problem specific naming. Clustering problem has a wide range of domains that it can be used, because it is useful in the exploration of data in data analysis. Basic idea of data clustering is to group similar objects into the same cluster. From this definition, we can understand that items that are in different clusters are dissimilar with each other.

Data Clustering techniques require a definition of a similarity measure between patterns, which is not easy to specify in the absence of any prior knowledge about the clusters. Therefore, even there exists lots of clustering algorithms, none of them can handle

all sorts of clustering problems. Each algorithm has its own approach for handling cluster validity, number of clusters and structure of the data.

In [60] clustering approaches are classified as *hierarchical* and *partitional*. Hierarchical approaches includes single link and multi link, on the other hand partitional approaches include square error, k-means, graph theoretic, mixture resolving, expectation minimization and mode seeking. LLE is used in a novel clustering approach that devises a genetic algorithm in [53].

3.4. EXAM TIMETABLING

An optimal schedule is searched with a set of constraints for a given set of events and resources in timetabling problems. There are two types of constraints which are *hard constraints* and *soft constraints*. Hard constraints have to be satisfied to solve the scheduling problem, on the other hand soft constraints are not mandatory to solve but the aim is to resolve soft constraints as many as possible. A solution must not violate any hard constraints to be named a feasible solution. The size of the search space changes. Number of items to be scheduled changes the size of the search space for a timetabling problem. Constraints result many *infeasible* solutions in the search space.

[61] and [62] are the first studies with some computer based strategies for examination timetabling. Then, a large university examination timetabling system is designed by [63]. Real-world applications of timetabling in different universities are provided as a survey in [64]. In this study how to design specific timetabling algorithms for each institution separately is described. Different heuristic orderings based on graph coloring applied in [5]. This approach takes its inspiration from [65] which showed that the timetabling problem can be reduced to a graph coloring problem. [66] proposed an effective use of evolutionary algorithms by dividing a large scale problem into smaller instances and solving these instances separately. [67] designed a multi-objective evolutionary algorithm(MOEA) based on a direct encoding of the mapping between exams and time slots. The approach attempted to minimize the number of violations of each type of constraint as separate objectives. [68] proposed an XML data format which is based on MathML for representing

timetabling problems and their solutions. [28] tested a set of hyper-heuristics that combine heuristic selection and move acceptance mechanisms over a set of examination timetabling benchmark problems. [69] presented a multi-objective evolutionary algorithm(MOEA) that aims to generate feasible exam timetables without any prior knowledge of timetable length.

3.5. GRAPH COLORING

Graph Coloring Problem (GCP) is a well known combinatorial optimization problem which is proved to be NP Complete in [70]. Graph coloring is assigning colors to each vertex of an undirected graph such that no adjacent vertices should receive the same color. The minimum number of colors that can be used for a valid coloring is called the *chromatic number*.

Given a graph $G = (V, E)$ with vertex set V and edge set E , and given an integer k , a k -coloring of G is a function $c : V \rightarrow 1, \dots, k$. The value of $c(x)$ of a vertex is called the color of x . The vertices with color r ($1 \leq r \leq k$) define a color class, denoted V_r . If two adjacent vertices x and y have the same color r , x and y are conflicting vertices, and the edge (x, y) is called a conflicting edge. If there is no conflicting edge, then the color classes are all independent sets and the k -coloring is valid. The Graph Coloring Problem is to determine the minimum integer k (the chromatic number of $G - x(G)$) such that there exists a legal k -coloring of G [71]

There are several studies about Graph Coloring Problem. Recursive Largest Fit (RLF) is a well-known greedy heuristic introduced by Leighton [65]. Davis [72] proposed a coding as an ordering of vertices which could be used in a genetic algorithm. Davis' algorithm was designed to maximize the total weights of the vertices in the graph colored with a fixed amount of colors. Hertz and de Werra [73] presented the first tabu search implementation which outperforms another local search method, simulated annealing on random dense graph instances. This methods attempts to find a valid k -coloring by partitioning the set of vertices V into k subsets. [74] presented three simulated annealing implementations based on three neighboring approaches: penalty-function approach, Kempe chain approach and fixed-k approach. [71] proposed a variable neighborhood search algorithm(VNS) for graph coloring

problem.

3.6. BIN PACKING

Bin packing problem(BPP) is a well-known combinatorial optimization problem which is NP hard problem. In BPP items of different sizes has to be packed into a minimal number of bins of fixed capacity. Bin packing has many variants with respect to:

- the number of dimensions used
- the arrival of bins
- the distribution of the size of the items

In this study, only one dimensional bin packing problem will be considered.

In the classic one dimensional bin packing problem [75], there is a given sequence $L = (a_1, a_2, a_3, \dots, a_n)$ of items, each with a size $s(a_i) \in (0, 1]$ and these are aimed to be packed into a number of bins which are in a fixed unit capacity. In other words the aim of the problem is to partition the given items into a minimum number of subsets. Let's say the number of bins is m and the subsets are $B_1, B_2, B_3, \dots, B_m$, this could be formulated as in 3.2

$$\sum_{a_i \in B_j} \leq 1, 1 \leq j \leq m \quad (3.2)$$

There are several methods to solve bin packing problem. *First Fit* is one of them. In this approach an item a_i is placed in the first partially filled bin B_j into which it fits. If there is no bin that a_i can fit into a new bin is created and this item is put into this new bin. This method is used in this study while initializing the Bin Packing problem. Another method to solve bin packing problem is *Best Fit*. In this approach an item a_i is placed in the partially filled bin B_j with the highest level which means the selected bin is the most suitable bin to put the item a_i in. On the other hand *Worst Fit* is the counter approach of best fit and it places an item a_i in the partially filled bin B_j with the lowest level.

Most comparative studies of bin packing take the Martello and Toth's branch-and-bound reduction algorithm(MTP) [76] as their reference. The MTP is based on a dominance criterion. A feasible set of items is defined as any subset $F \subseteq N$ such that $\sum_{i \in F} w_i \leq C$. With two feasible sets F_1 and F_2 , F_1 dominates F_2 if and only if the number of bins in some optimal solution by setting $B_1 = F_1$ is not greater than by setting $B_1 = F_2$. There exists a partition P_1, \dots, P_t and a subset $i_1, \dots, i_l \subset F_1$ such that $w_{ih} \geq \sum_{k \in P_h} w_k$ for $h = 1, \dots, l$. Therefore a solution containing F_1 will not have more bins than a solution containing F_2

[7] uses a Hybrid Grouping Genetic Algorithm(HGGA) which is heavily modified to suit the structure of grouping problems. While placing free items Falkenauer used a similar strategy like domination criterion of Martello and Toth.

4. SELECTION HYPERHEURISTICS FOR GROUPING PROBLEMS

Heuristic is a methodology to solve problems where exhaustive solutions aren't practical. A heuristic does not solve all problems at best every time. Heuristics are problem dependent solution methods, so heuristics are implemented separately for every different problem domain and/or problem instance. A type of a hyper-heuristic can be described as a heuristic to choose heuristics [12]. A selection hyper-heuristic operates at a high level and manages a set of low level heuristics (see Section 2.1.3). The number of low level heuristics are not limited. New heuristics can be implemented and added into the hyper-heuristic framework.

A selection hyper-heuristic framework mainly combines two components:

- The *heuristic selection* method is a problem-independent structure to make selection of one of the low-level heuristics provided to apply on the problem.
- After the heuristic is applied, the other important part of the framework, the *move acceptance* mechanism gathers the candidate solution in terms of a fitness value and decides to accept or reject the new fitness value returned by the applied heuristic.

The following sections describe all relevant components of the hyper-heuristic framework for solving grouping problems.

4.1. FITNESS EVALUATION

4.1.1. Graph Coloring and Exam Timetabling

In [6], it is 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. Therefore in this study, they are treated the same when solving them and their fitness functions are calculated in the same way. The fitness value denotes the number of

conflicts in the graph. The algorithm works on the principle of counting the adjacent nodes of the graph. If there are two adjacent nodes that are the same color, this counts as a conflict.

4.1.2. Data Clustering

In Data Clustering, fitness function calculates the total distance of items in clusters. Euclidean distance metric is used in this research. In Euclidean metric, there are n centers for each group where n is the number of properties. Therefore every property has its own center and the distance of each item is calculated with the distances to these property centers. It is formulated as in Equation 4.1

$$\sum_{k=1}^n \sum_{i=1}^m W_{ik} \sum_{j=1}^p (X_{ij} - C_{kj})^2 \quad (4.1)$$

In Equation 4.1 n is the number of clusters, m is the number of items, p is the number of properties and W_{ik} is 1 if the i^{th} item is in k^{th} cluster, it is 0 otherwise. C_{kj} is the center for the k^{th} clusters j^{th} property, while X_{ij} is the i^{th} items j^{th} property.

4.1.3. Bin Packing

There are different approaches while calculating the fitness value of the bin packing problems. A straightforward fitness function would just take the inverse of the number of bins. However, as pointed by Falkenauer in [77] as well, such a fitness function will result a very unfriendly fitness landscape in which many combinations with one more bin than optimal solution will have the same fitness value. Instead the function proposed by [77] is used in this study:

$$f(s) = \frac{\sum_{i=1}^N (F_i/C)^2}{N} \quad (4.2)$$

where, N is the number of bins, F_i is the fill of the bin i and C is maximum bin capacity for a given solution s . This equation is also used in [59], the results of this study and [59] will be compared.

4.1.4. Delta Fitness Evaluation

Grouping hyper-heuristics work on time based fashion, all runs are executed in a predefined time. Fitness evaluation is a very time consuming part of the framework. In almost all grouping hyper-heuristics, the fitness evaluation is done based on the whole candidate solutions. The datasets can be very large in grouping problems, therefore calculating the fitness value of a candidate solution takes significant time. Number of iterations is an important factor in finding the best solution, the more iterations in a run increases the probability of finding the best solution for the problem instance. This leads us to utilize a different fitness calculation which is called *delta fitness evaluation*. In delta fitness evaluation, rather than calculating the fitness value of the whole candidate solution, only the partial fitness contributions of the groups that are modified during that iteration are calculated. This would give a significant time advantage and provides more iterations in a run. For example, if swap heuristic is applied and G_1 and G_2 are the groups that swap nodes n_i and n_j , it is calculated as in equation 4.3

$$fitness = (fitness - (f(G_1) + f(G_2))) + (f(G'_1) + f(G'_2)) \quad (4.3)$$

To generalize the equation, the partial fitness contribution (denoted as $f(.)$) of the groups that will be modified are subtracted from the overall fitness value and the new partial fitness contributions are calculated after the low-level heuristic is applied to the candidate solution and these are added to the fitness value of the candidate solution.

4.2. HEURISTIC SELECTION METHODS

Despite the difference, single-objective approaches deal with only the best and unique solution, the multi-objective optimization approaches deal with conflicting objectives with a set of solutions (pareto optimal set) instead of only one. So that it is not possible to expect the same selection procedures within the single-objective optimization to be applicable on multi-objective optimization problems. For this study, we implemented three heuristic selection mechanism namely “Simple Random”, “Reinforcement Learning” and “Modified

Reinforcement Learning”.

4.2.1. Simple Random

“Simple Random” selection mechanism makes the selection randomly among the low-level heuristics. It is totally random, no intelligence used in this selection mechanism and every heuristic has an equal chance to be selected in every single run.

4.2.2. Reinforcement Learning

“Reinforcement Learning” selection mechanism makes the selection based on a ranking mechanism. Every heuristic has a score in reinforcement learning algorithm and they all start equally. The score of heuristics are bounded with a range and they are always in this score range. There are studies about the range of these bounds. [78] showed that the upper and lower bounds should be 0 and 40. These values are used as the bounds of reinforcement learning in this study. When heuristics are selected and makes an improvement in that run they take a reward which increases their score. However when a heuristic is applied but it is not improve the candidate solution, it is given a punishment score which decreases its score. Reinforcement learning selects the heuristics based on their scores. Therefore all heuristics do not have the same probability to be selected for the next run. If all the heuristics have the same score than a random heuristic is selected to apply on the candidate solution. This case is same as the simple random selection mechanism. Hyper-heuristic framework continues to select a heuristic while it is successful to optimize the fitness value of the candidate solution. Algorithm 4.1 shows how the heuristic selection in reinforcement learning works and algorithm 4.2 shows how the scores of heuristics change in reinforcement learning.


```

1: indexArray[numberOfHeuristics]
2: scoreArray[numberOfHeuristics]
3: counter ← 1
4: indexArray[0] ← 0
5: max ← scoreArray[0]
6: for index = 1 to numberOfHeuristics do
7:   if scoreArray[index] > max then
8:     max ← scoreArray[index]
9:     indexArray[0] ← index
10:    counter ← 1
11:   else
12:     indexArray[counter + +] ← index
13:   end if
14: end for
15: if counter = 1 then
16:   tempIndex ← indexArray[0]
17: else
18:   tempIndex ← indexArray[random(counter)]
19: end if
20: return tempIndex

```

Figure 4.1. Reinforcement Learning algorithm choose heuristic procedure

```

1: if  $newFitness < oldFitness$  then
2:    $score[selectedHeuristicIndex] + = reward$ 
3: else
4:    $score[selectedHeuristicIndex] - = punishment$ 
5: end if
6: if  $score[selectedHeuristicIndex] > scoreUpperBound$  then
7:    $score[selectedHeuristicIndex] = scoreUpperBound$ 
8: end if
9: if  $score[selectedHeuristicIndex] < scoreLowerBound$  then
10:   $score[selectedHeuristicIndex] = scoreLowerBound$ 
11: end if

```

Figure 4.2. Reinforcement Learning algorithm change score procedure

4.2.3. Modified Reinforcement Learning

Modified Reinforcement Learning is a variant of reinforcement learning which is also based on a similar scoring mechanism. Modified reinforcement learning differs from reinforcement learning in changing the scores of the heuristics. As stated before reinforcement learning changes the scores of the heuristics based on the improvement in the candidate solution. Modified reinforcement learning adds an extra scoring mechanism to this and it changes the scores of the heuristics based on their acceptance. If a heuristic is accepted by the acceptance criteria, it is given an extra reward score that increases its score and it does not change the heuristic's score if it is not accepted. The heuristic selection methodology is completely the same with its original version of reinforcement learning. The modified scoring mechanism of modified reinforcement learning is in algorithm 4.3.

```

1: if  $newFitness < oldFitness$  then
2:    $score[selectedHeuristicIndex] + = reward$ 
3: else
4:    $score[selectedHeuristicIndex] - = punishment$ 
5: end if
6: if Selected heuristic is accepted then
7:    $score[selectedHeuristicIndex] + = reward$ 
8: end if
9: if  $score[selectedHeuristicIndex] > scoreUpperBound$  then
10:   $score[selectedHeuristicIndex] = scoreUpperBound$ 
11: end if
12: if  $score[selectedHeuristicIndex] < scoreLowerBound$  then
13:   $score[selectedHeuristicIndex] = scoreLowerBound$ 
14: end if

```

Figure 4.3. Modified Reinforcement Learning algorithm change score procedure

4.3. MOVE ACCEPTANCE METHODS

The combination of heuristic selection methods and the move acceptance methods is an important issue for performing better results on various objective functions. An important observation in various studies is that the performance difference between various acceptance methods gives significant changes in comparison to the performance differences of applied selection methods. This gives the idea that the importance of the move acceptance method used in hyper-heuristic framework makes more significance on the performance rather than the selection method. In this study, we applied three acceptance mechanism namely; Improving or Equal (IEQ), Great Deluge(GDEL) and Late Acceptance(LACC).

4.3.1. Great Deluge

Great Deluge (GDEL) acceptance criterion is a stochastic acceptance method. All moves that improve the fitness value are accepted. On the other hand, the moves that doesn't improve the fitness value are accepted according to their objective value. If it is better than the expected objective value, the worsening move is accepted. Worsening moves probabilities to be accepted are higher in the early stages of the run and their probability decreases with a linear rate. This can be formulated as in Equation 4.4

$$\tau_t = f_0 + \Delta F \times \left(1 - \frac{t}{T}\right) \quad (4.4)$$

τ_t is the threshold level at step t in a minimization problem, T is the maximum number of steps, ΔF is an expected range for the maximum fitness change and f_0 is the final objective value. In the implementation, algorithm 4.4 shows how it is initialized and algorithm 4.5 shows Great Deluge's acceptance mechanism. The acceptance probability of worsening moves decreases linearly by time.

```

1: minFitness ← 10000
2: startTime ← currentTime
3: for candidateSolutionIndex ← 0 to numberOfCandidateSolutions do
4:   if fitness(candidateSolutions[candidateSolutionIndex]) < minFitness then
5:     minFitness = fitness(candidateSolutions[candidateSolutionIndex])
6:   end if
7: end for
8: cnorm = minFitness

```

Figure 4.4. Great Deluge algorithm initialization procedure

```

1: if newFitness < currentFitnessValueOfCandidateSolution then
2:   return true
3: else
4:   time = currentTime – startTime
5:   delta = 1.0 – time/durationOfATrial
6:   if delta ≤ 0.0 then
7:     delta = 0.0
8:   end if
9:   if fitness < globalOptimum + cnorm * delta then
10:    return true
11:  end if
12: end if
13: return false

```

Figure 4.5. Great Deluge algorithm accept procedure

4.3.2. Improve or Equal

“Improve or Equal”(IEQ) acceptance mechanism is a deterministic and efficient method. As its name implies it accepts moves that improves the candidate solution or moves that doesn’t change the fitness value of the candidate solution.

4.3.3. Late Acceptance

“Late Acceptance”(LACC) is a newly proposed meta-heuristic acceptance mechanism, it is firstly proposed by [57]. Late Acceptance is some kind of iterative search technique, however it uses a different acceptance mechanism. It chooses to compare new candidate solution with a previous one instead of current candidate solution. As it is observed in initialization part in the algorithm 4.6 there is a list of previous solutions. Then in algorithm 4.7, it is observed that the new candidate solution is compared with a previous one from the list.

```

1: lastBestFitness[numberOfCandidateSolutions]
2: fitnessArray[numberOfCandidateSolutions]
3: for candidateSolutionIndex  $\leftarrow$  0 to numberOfCandidateSolutions do
4:   for listIndex  $\leftarrow$  0 to listSize do
5:     fitnessArray[candidateSolutionIndex][listIndex]  $\leftarrow$ 
       fitnessOfCandidateSolution
6:   end for
7: end for
8: vhead  $\leftarrow$  0

```

Figure 4.6. Late Acceptance algorithm initialization procedure

```

1: candidateSolutionIndex  $\leftarrow$  activeCandidateSolutionIndex
2: retValue  $\leftarrow$  false
3: if newFitness < fitnessArray[candidateSolutionIndex][vhead] then
4:   retValue  $\leftarrow$  true
5:   fitnessArray[candidateSolutionIndex][vhead]  $\leftarrow$  newFitness
6:   lastBestFitness[candidateSolutionIndex]  $\leftarrow$  newFitness
7: else
8:   retValue  $\leftarrow$  false
9:   fitnessArray[candidateSolutionIndex][vhead]  $\leftarrow$ 
       lastBestFitness[candidateSolutionIndex]
10: end if
11: vhead++
12: if vhead > listsize - 1 then
13:   vhead  $\leftarrow$  0
14: end if
15: return retValue

```

Figure 4.7. Late Acceptance algorithm accept procedure

4.4. LOW LEVEL HEURISTICS

There are 10 low level heuristics used in this study. These are namely Swap, Divide, Divide Tournament, Divide Most Conflicting, Merge, Merge Tournament, Merge Most Conflicting, Change, Change from Most Conflicting to Most Suitable and Change with Most Suitable. One of the main aims of the study is to solve all problems using same low level heuristics and hence the same hyper-heuristic framework, therefore very simple and basic low-level heuristics are used.

All heuristics first chooses a point from the list of non-dominating candidate solutions and checks whether the random point is in the range of the non-dominating candidate solutions list. The main algorithm requires a range for the list of non-dominating candidate solutions. The best solutions of the problems are known, therefore we specify the range of the non-dominating candidate solutions list based on these best solutions. Different ranges are tested before we ensure which range fits best for the list of non-dominating candidate solutions. Some best solutions of these problems, mostly the data clustering problem's best solutions are relatively small numbers therefore we can not have equal plus and minus range for these problems. If the best solutions permits the equal plus-minus range, we give them an equal plus-minus range. For example, if the best solution of the problem is 15, we give the range between 5 to 25 or 10 to 20 based on our interval parameter. However if the best solution is 3, then we give the range between 2 to 10 or 2 to 8 based on our interval parameter. If we have a list of non-dominating candidate solutions of 10 to 20 range, this means that we have candidate solutions which have at least 10 groups and which have at most 20 groups in the solutions.

In our heuristic set we have 3 algorithms that decreases the number of groups, 3 algorithms that increases the number of groups and 4 algorithms that doesn't change the number of groups. The divide algorithms increases the number of groups and merge algorithms decreases the number of groups, therefore before implementing the algorithms, a range check is implemented to ensure this rule. If we implement the merge heuristics, which will decrease the number of groups, we can not choose the candidate solution with the least number of groups. For example, if we have the range of 10 to 20 we can not choose

the candidate solution representation with 10 groups, because after we implement the merge heuristics the number of groups will be 9 and we do not have any candidate solution that we can compare with this number of groups. In the divide heuristics case, we increase the number of groups by one. Therefore if we choose the solution with 20 groups, after we implement one of the divide heuristics the number of groups will be 21 and there are no candidate solutions that we can compare with this number of groups. This takes us to restrict the hyper-heuristic not to choose the most number of groups solution if we choose one of the divide heuristics to apply. In our example, we can not choose the candidate solution with 20 groups to apply divide heuristics. Here are the details of the low-level heuristics algorithms.

4.4.1. Swap

Swap heuristic is one of the heuristics that doesn't change the number of groups in the solution. Main idea of the swap heuristic is to swap two nodes in two different groups. Swap heuristic chooses a candidate solution from the list of non-dominating candidate solutions to apply. Then it selects two random groups from the candidate solution and it chooses two random nodes from both groups. Take these nodes out of these groups and put them in other groups that are chosen before. It is a mutational heuristic works for exploration of the search space. The heuristic is illustrated in Figure 4.9.

- 1: Select a candidate solution (from a list of non-dominating candidate solutions)
- 2: Select a random group(G_1)
- 3: Select a random group(G_2 , where $G_2 \neq G_1$)
- 4: Select a random node(N_1) from G_1
- 5: Select a random node(N_2) from G_2
- 6: Remove N_1 from G_1 and N_2 from G_2
- 7: Place N_1 to G_2 and N_2 to G_1
- 8: Calculate fitness based on delta evaluation using G_1 and G_2

Figure 4.8. Swap algorithm

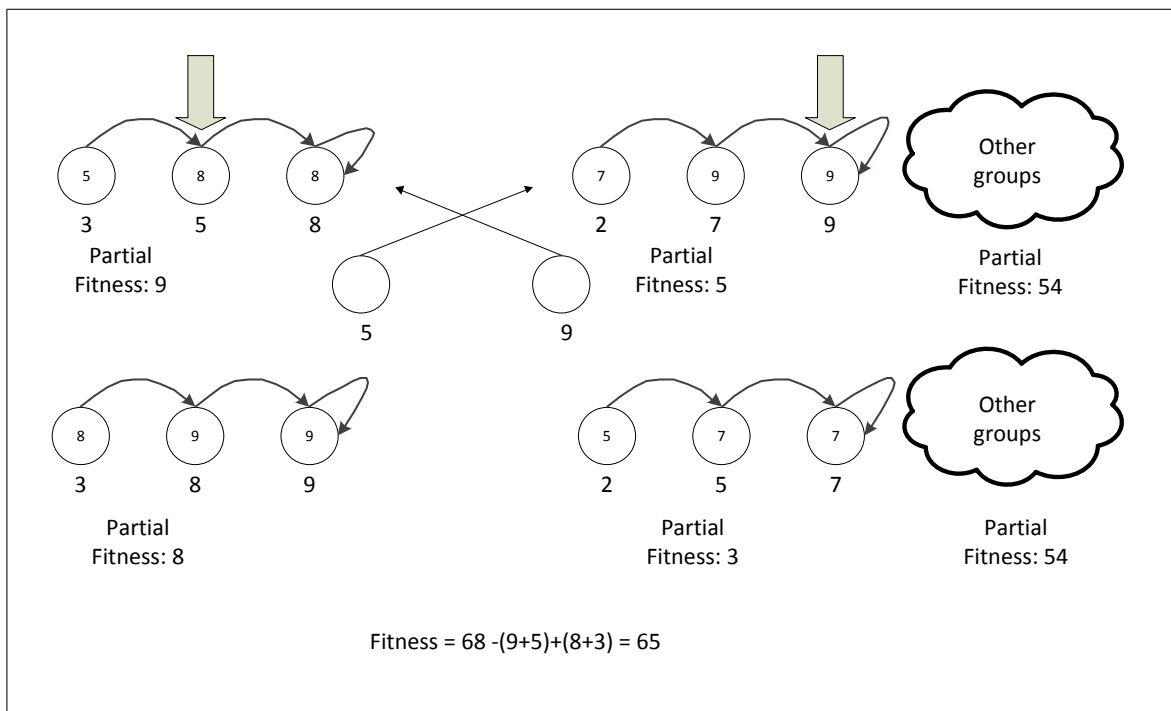


Figure 4.9. Swap Low-level Heuristic

4.4.2. Merge

Merge heuristic decreases the number of groups in a candidate solution by merging two groups. There is a probability to increase the total error rate but it is useful for especially in the cases which the items of the same group are separated into two different groups.

Merge heuristic chooses a candidate solution from a list of non-dominating candidate solutions to apply. Then it selects two random groups from the candidate solution and merges these two groups. After merging two groups a repair mechanism is applied to ensure the LLE structure is assured. Merge heuristic must choose a candidate solution that does not violate the range of the list of non-dominating candidate solutions. It means that the candidate solution with the smallest number of groups can not be chosen for merge heuristic. The heuristic is illustrated in Figure 4.11.

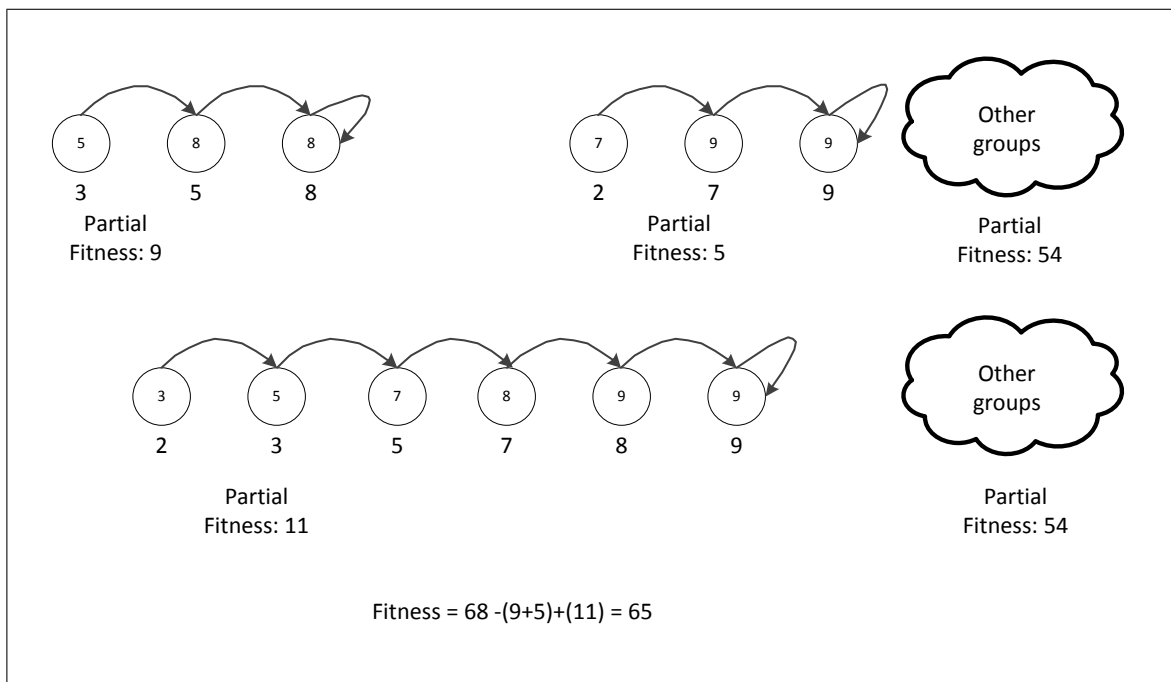


Figure 4.11. Merge Low-level Heuristic

- 1: Select a candidate solution (from a list of non-dominating candidate solutions)
- 2: Select a random group(G_1)
- 3: Select a random group(G_2 , where $G_2 \neq G_1$)
- 4: Merge G_1 and $G_2 \rightarrow G_3$
- 5: Repair G_3 to assure LLE structure
- 6: Calculate fitness based on delta evaluation using G_1 , G_2 and G_3

Figure 4.10. Merge algorithm

4.4.3. Merge Tournament

Merge tournament is a different version of merge heuristic. It also decreases the number of groups in a candidate solution by merging two groups and there is possibility of increasing the overall fitness but it is useful for especially the cases which the items of same group are separated into two different groups. Also merge tournament as in merge heuristic must obey the range of the list of non-dominating candidate solutions.

Merge tournament differs from merge heuristic by its tournament structure. Tournament procedure is based on the partial fitness contribution of each group to the overall fitness value for a given candidate solution. Tournament size is a parameter for merge tournament which indicates how many groups are challenging in the tournament. Different experiments run for different number of tournament sizes. For example if the tournament size is 2, then two randomly selected groups are competing. The group with the highest partial fitness contribution wins the tournament and it is selected. Therefore in merge tournament the groups are selected based on this tournament procedure. In Figure 4.20, the tournament procedure is provided. The heuristic is illustrated in Figure 4.13.

- 1: Select a candidate solution (from a list of non-dominating candidate solutions)
- 2: Select a group using tournament(G_1)
- 3: Select a group using tournament(G_2 , where $G_2 \neq G_1$)
- 4: Merge G_1 and $G_2 \rightarrow G_3$
- 5: Repair G_3 to assure LLE structure
- 6: Calculate fitness based on delta evaluation using G_1 , G_2 and G_3

Figure 4.12. Merge Tournament algorithm

4.4.4. Merge Most Conflicting

Merge Most Conflicting (MergeMC) heuristic is a variant of the merge heuristic. As in all types of merge heuristics, it decreases the number of groups. In merge heuristic, the groups to be merged are selected randomly, on the other hand in MergeMC the groups are selected based on their partial fitness contributions. If $partialFitnessContribution(G_i) > partialFitnessContribution(G_j)$ that means that G_i has more conflict than G_j . The procedure represented in algorithm 4.14 finds the most conflicting group in the candidate solution to use in the heuristic. MergeMC chooses both of its groups based on this procedure. The heuristic is illustrated in Figure 4.16.

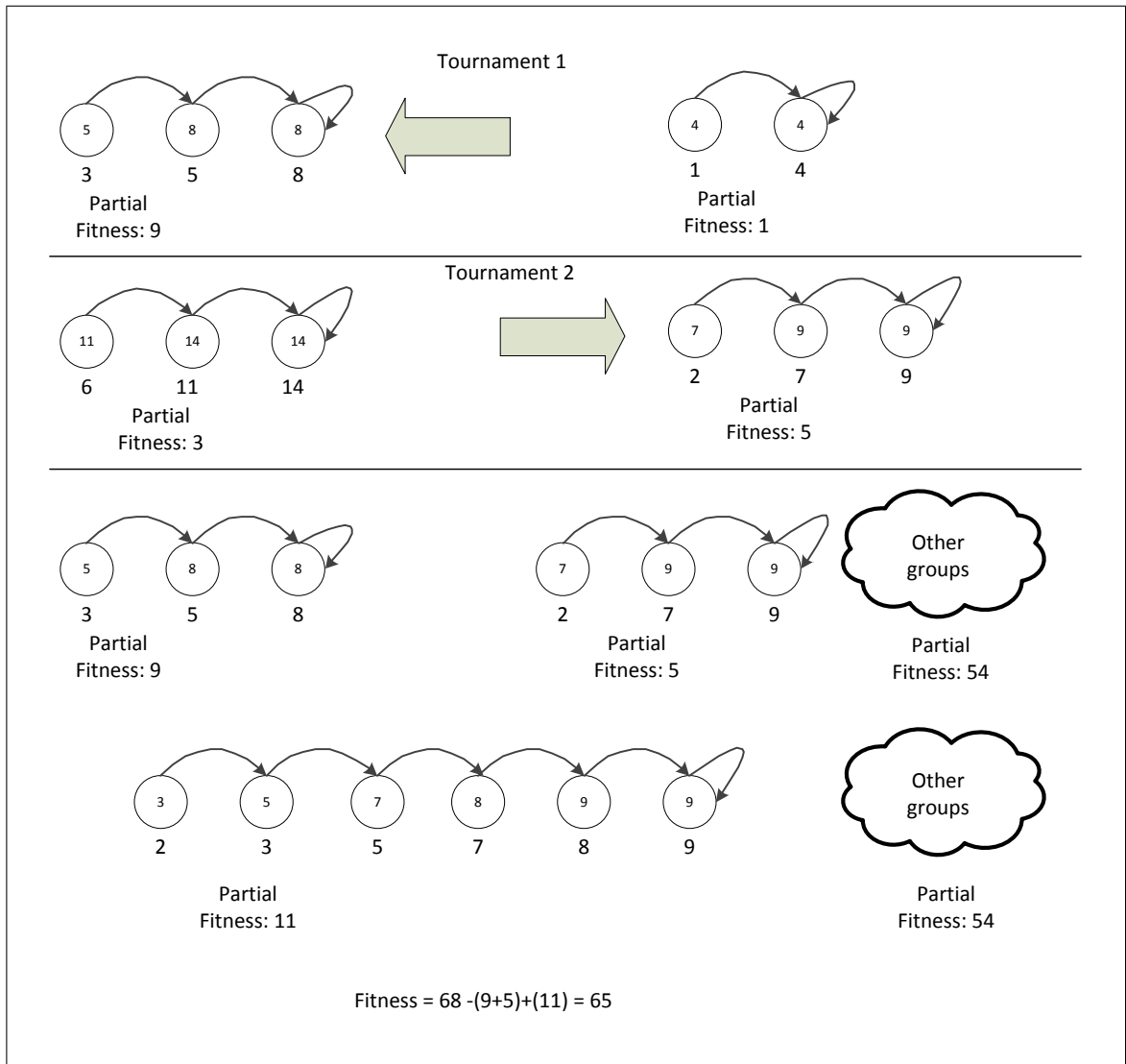


Figure 4.13. Merge Tournament Low-level Heuristic

```

1:  $groupIndex \leftarrow -1$ 
2:  $tempFitnessContribution \leftarrow 0$ 
3: for  $index \leftarrow 0$  to  $candidateSolution.groupSize$  do
4:   if  $tempFitnessContribution < group.partialFitnessContribution$  then
5:      $groupIndex = index$ 
6:      $tempFitnessContribution = group.partialFitnessContribution$ 
7:   end if
8: end for
9: return  $groupIndex$ 

```

Figure 4.14. Finding Most Conflicting Group

```

1: Select a candidate solution (from a list of non-dominating candidate solutions)
2: Select the most conflicting group(G1)
3: Remove G1 from candidate solution
4: Select the most conflicting group(G2 , where  $G2 \neq G1$ )
5: Remove G2 from candidate solution
6: Merge  $G1$  and  $G2 \rightarrow G3$ 
7: Repair G3 to assure LLE structure
8: Calculate fitness based on delta evaluation using G1, G2 and G3

```

Figure 4.15. Merge Most Conflicting algorithm

4.4.5. Divide

Divide heuristic, as the name implies divides a group into two different groups. It increases the number of clusters of a candidate solution but meanwhile it decreases the error rate. It is useful when the items belonging to more than one groups are assigned into one group. Divide heuristic must assure the range of the list of non-dominating candidate solutions. The candidate solution with most number of groups can not be selected for divide

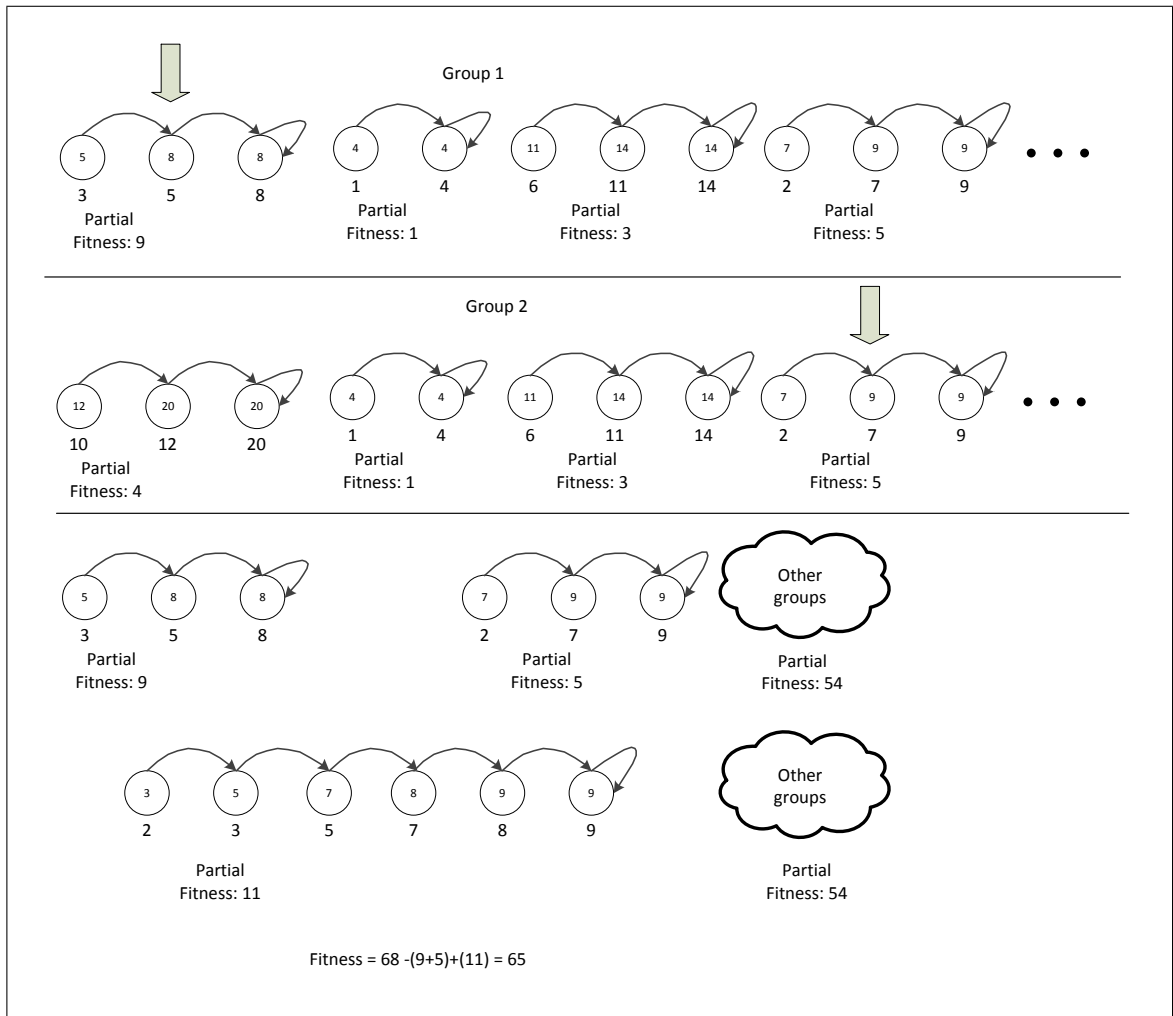


Figure 4.16. Merge Most Conflicting Low-level Heuristic

heuristic, because when we apply divide heuristic, number of groups increases one and it violates the range of the list of non-dominating candidate solutions. A random group is chosen from the selected candidate solution to apply the divide heuristic and a random node is selected from this group as the division point in the group. The heuristic is illustrated in Figure 4.18.

```

1: Select a candidate solution (from a list of non-dominating candidate solutions)
2: Select a random group (G1)
3: if  $numOfNodesG1 \leq 1$  then
4:   Select another group for G1
5: end if
6: Divide  $G1 \rightarrow G2 - G3$ 
7: Calculate fitness based on delta evaluation using G1, G2 and G3

```

Figure 4.17. Divide algorithm

4.4.6. Divide Tournament

Divide tournament is a different implementation of divide heuristic. Divide tournament uses a tournament procedure as in merge tournament heuristic. It increases the number of heuristic and the range of the list of non-dominating candidate solutions must be assured while applying this heuristic. The candidate solution with most number of groups can not be selected to apply divide tournament heuristic.

Tournament procedure is same as the merge tournament, different number of groups compete for selection. The tournament size is a parameter to the algorithm and it is tested for different numbers. The group with the highest partial fitness contribution is chosen to divide. Highest partial fitness contribution implies that it has the highest number of conflicts in the group compared to other groups in the tournament. The implementation of the divide tournament is same of divide heuristic other than the tournament procedure. In algorithm 4.20, tournament procedure is explained in detail. The heuristic is illustrated in Figure 4.21.

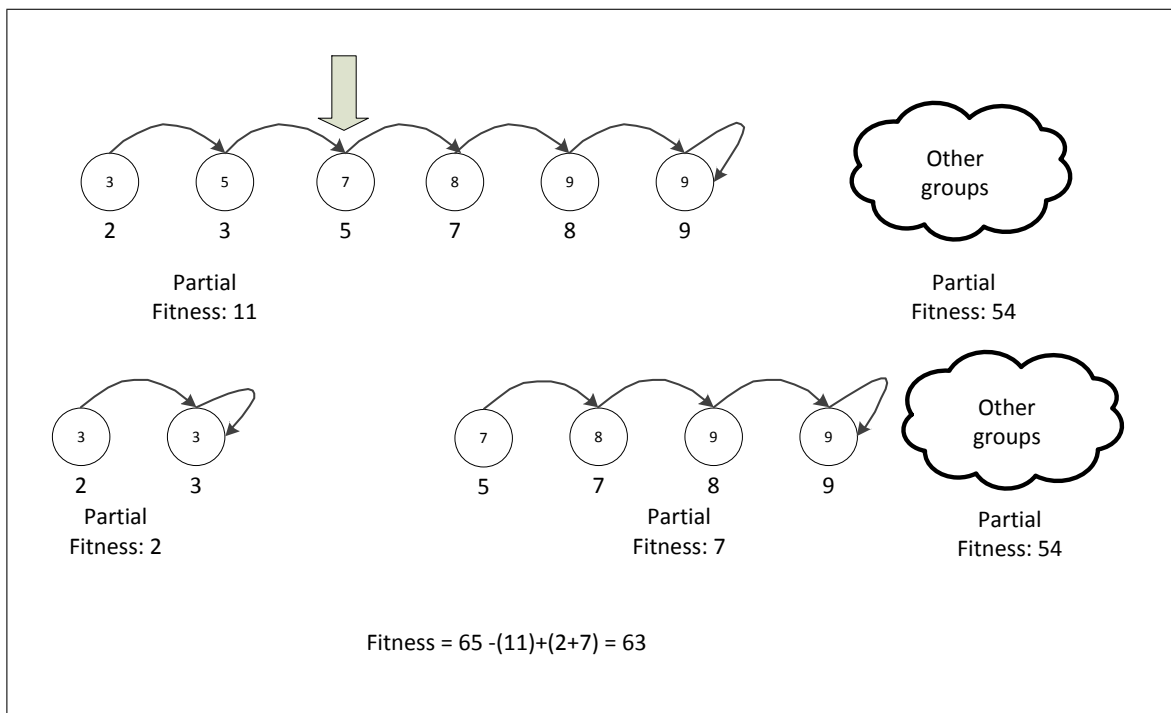


Figure 4.18. Divide Low-level Heuristic

- 1: Select a candidate solution (from a list of non-dominating candidate solutions)
- 2: Select a group by tournament ($G1$)
- 3: **if** $numOfNodesG1 \leq 1$ **then**
- 4: Select another group by tournament for $G1$
- 5: **end if**
- 6: Divide $G1 \rightarrow G2 - G3$
- 7: Calculate fitness based on delta evaluation using $G1$, $G2$ and $G3$

Figure 4.19. Divide Tournament algorithm


```

1:  $partialFitness \leftarrow 0$ 
2:  $groupIndex \leftarrow -1$ 
3: for  $i \leftarrow 1$  to  $tournamentSize$  do
4:   Select a random group G1
5:   if G1 in  $selectedGroups$  then
6:     Select another group for G1
7:   end if
8:   if  $partialFitness < partialFitnessOfSelectedGroup$  then
9:      $partialFitness = partialFitnessOfSelectedGroup$ 
10:     $groupIndex = indexOfSelectedGroup$ 
11:   else
12:     continue
13:   end if
14:   Add G1 to  $selectedGroups$ 
15: end for
16: return  $groupIndex$ 

```

Figure 4.20. Tournament algorithm

4.4.7. Divide Most Conflicting

Divide Most Conflicting(DivideMC) is a variant of divide heuristic. As all types of divide heuristic DivideMC increases the number of groups in the candidate solution. DivideMC differs from divide heuristic when choosing the group to divide. It chooses the group based on the *most conflicting* procedure which is described in algorithm 4.14 rather than choosing the group randomly. The heuristic is illustrated in Figure 4.23.

```

1: Select a candidate solution (from a list of non-dominating candidate solutions)
2: Select the most conflicting group (G1)
3: Divide  $G1 \rightarrow G2 - G3$ 
4: Calculate fitness based on delta evaluation using G1, G2 and G3

```

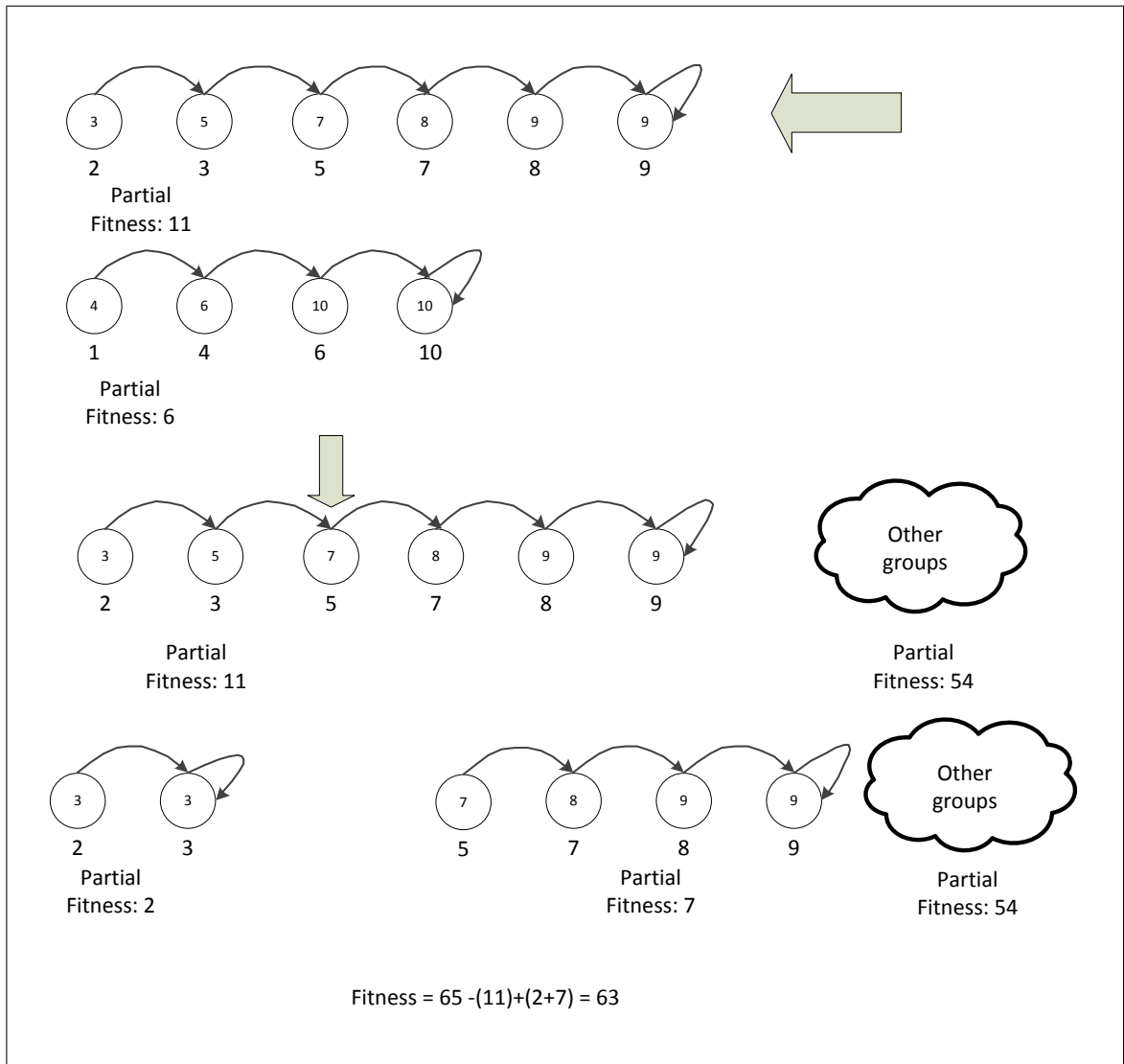


Figure 4.21. Divide Tournament Low-level Heuristic

Figure 4.22. Divide Most Conflicting algorithm

4.4.8. Change

Change heuristic is another heuristic that the number of groups doesn't change. Change heuristic selects a candidate solution from a list of non-dominating candidate solutions and then selects a group G_1 to take a node n_i from it and selects another group G_2 to put n_i into it. Change heuristic selects all groups and nodes randomly without having any special procedure. The heuristic is illustrated in Figure 4.25.

- 1: Select a candidate solution (from a list of non-dominating candidate solutions)
- 2: Select a random group(G_1)
- 3: Select a random group(G_2 , where $G_2 \neq G_1$)
- 4: Select a random node(N_1) from G_1
- 5: Remove N_1 from G_1
- 6: Place N_1 to G_2
- 7: Calculate fitness based on delta evaluation using G_1 and G_2

Figure 4.24. Change algorithm

4.4.9. Change from Most Conflicting to Most Suitable

Change from Most Conflicting to Most Suitable (CFMCTMS) heuristic is a variant of change heuristic. As in the change heuristic a candidate solution is chosen to apply the heuristic but differently from change heuristic CFMCTMS does not choose the groups randomly. CFMCTMS chooses the group G_1 to take the node out based on its conflicting level. Conflicting level represents the partial fitness contribution of the groups and CFMCTMS chooses the group with the highest partial fitness contribution to take the node out of it, this is represented in algorithm 4.14. After it chooses G_1 , it selects a random node

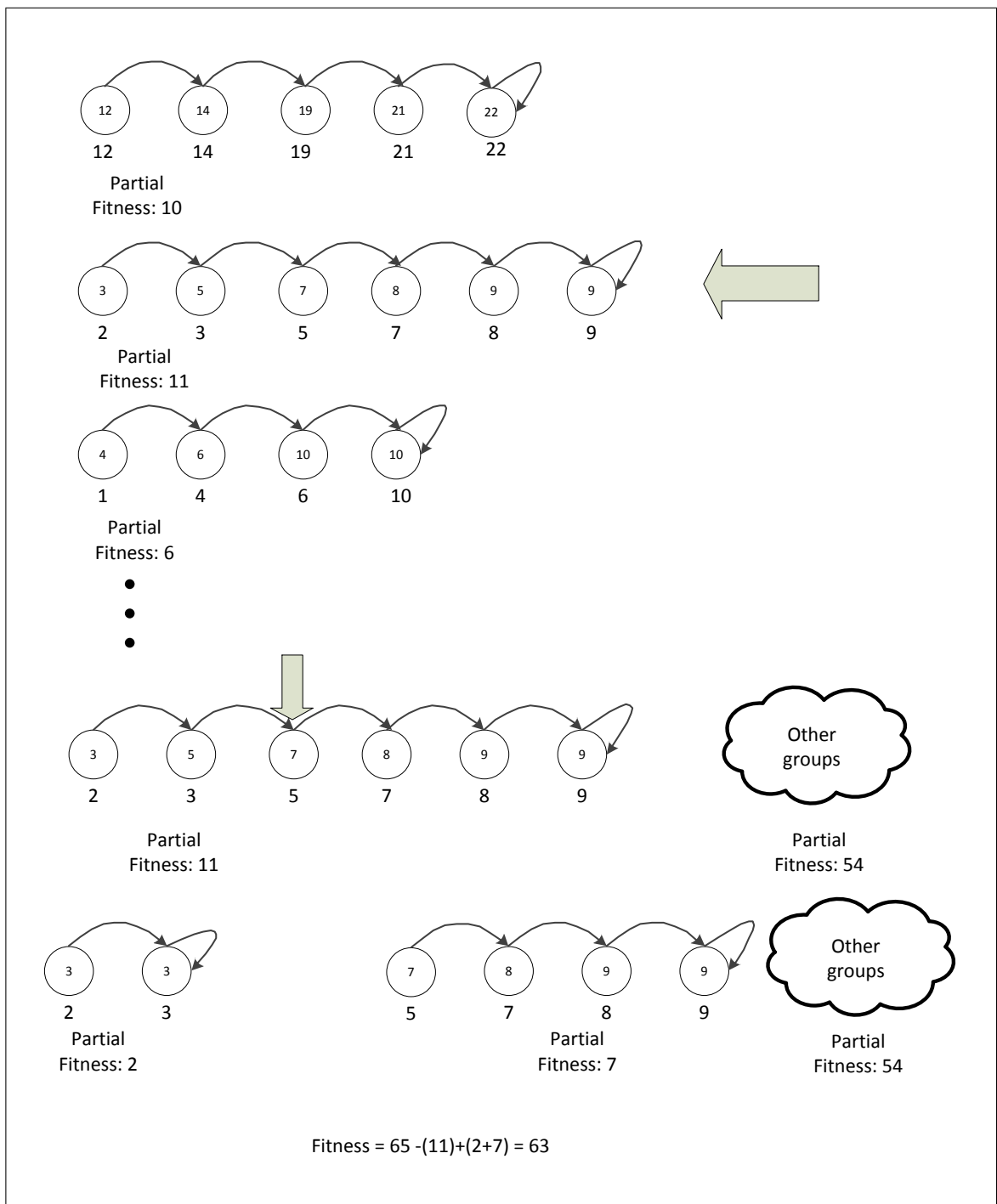


Figure 4.23. Divide Most Conflicting Low-level Heuristic

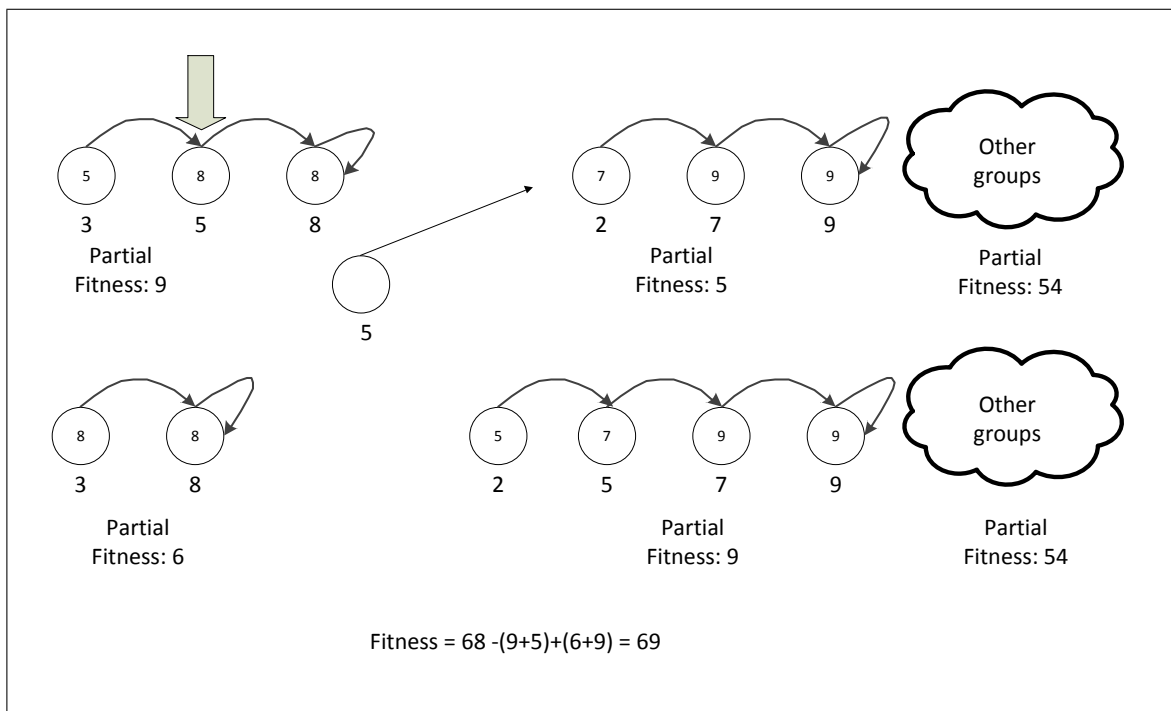


Figure 4.25. Change Low-level Heuristic

from that group and then searches for a group G_2 to place the selected node into. G_2 is also selected based on a defined procedure which selects the group according to its suitability. Best suitability is the best partial fitness contribution value after the node is placed on the group, this procedure is represented in algorithm 4.26. The heuristic is illustrated in Figure 4.28.

```

1:  $tempPartialFitness \leftarrow 100000$ 
2:  $groupToChose \leftarrow -1$ 
3: for  $groupIndex \leftarrow 0$  to  $candidateSolution.groupSize$  do
4:   Add new node to group  $G_{groupIndex}$ 
5:    $group.partialFitness \leftarrow newPartialFitnessOfTheGroup$ 
6:   if  $group.partialFitness < tempPartialFitness$  then
7:      $groupToChoose = groupIndex$ 
8:      $tempPartialFitness = group.partialFitness$ 
9:   end if
10: end for
11: return  $groupToChoose$ 

```

Figure 4.26. Finding Most Suitable Group

```

1: Select a candidate solution (from a list of non-dominating candidate solutions)
2: Select a most conflicting group( $G_1$ )
3: Select a random node( $N_1$ ) from  $G_1$ 
4: Remove  $N_1$  from  $G_1$ 
5: Select the most suitable group( $G_2$ )
6: Place  $N_1$  to  $G_2$ 
7: Calculate fitness based on delta evaluation using  $G_1$  and  $G_2$ 

```

Figure 4.27. Change from Most Conflicting to Most Suitable algorithm

4.4.10. Change with Most Suitable

Change with Most Suitable Heuristic(ChangeMS) is another variant of Change heuristic. It is different from Change and CFMCTMS, it chooses the group G_1 and node n_i randomly but chooses the group G_2 according to the most suitable procedure defined in algorithm 4.26. The heuristic is illustrated in Figure 4.30.

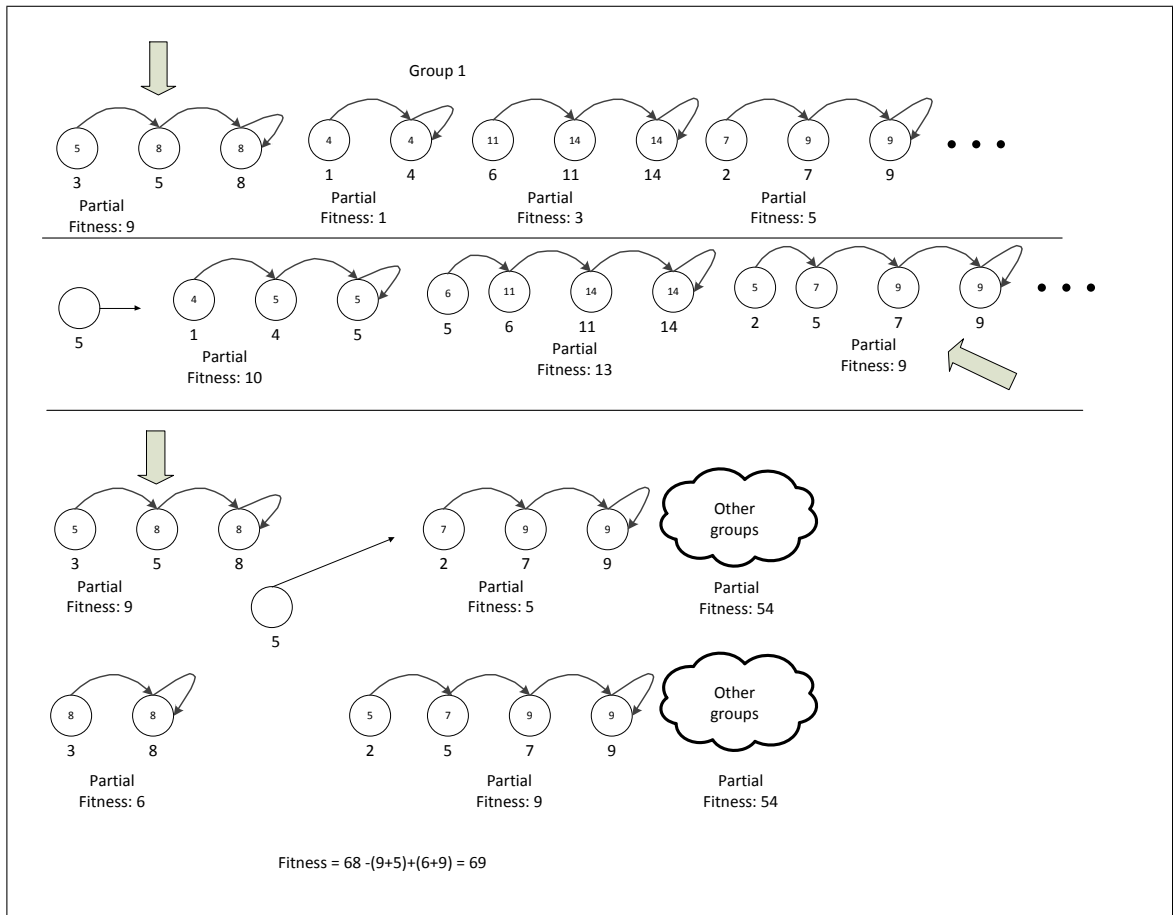


Figure 4.28. Change From Most Conflicting To Most Suitable Low-level Heuristic

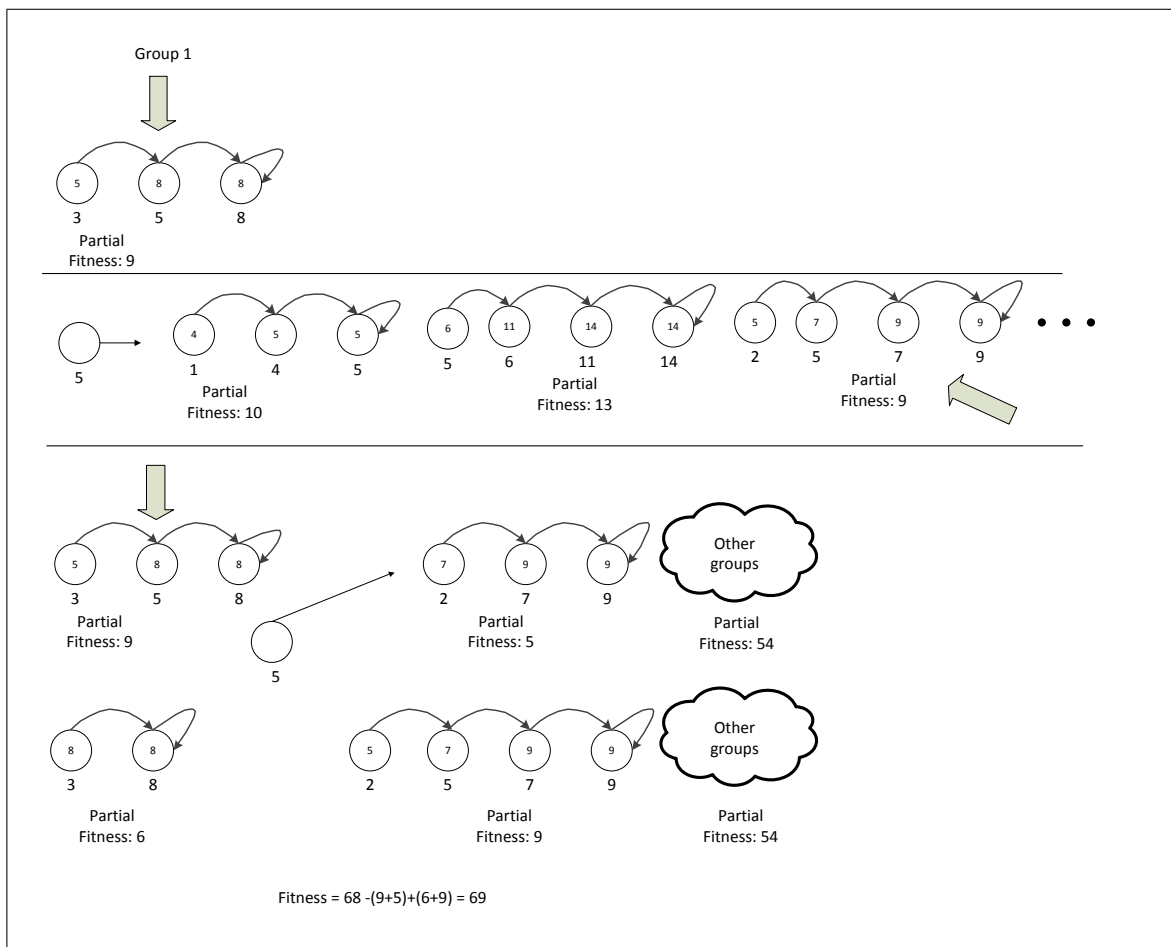


Figure 4.30. Change With Most Suitable Low-level Heuristic

- 1: Select a candidate solution (from a list of non-dominating candidate solutions)
- 2: Select a random group(G1)
- 3: Select a random node(N1) from G1
- 4: Remove N1 from G1
- 5: Select the most suitable group(G2)
- 6: Place N1 to G2
- 7: Calculate fitness based on delta evaluation using G1 and G2

Figure 4.29. Change with Most Suitable algorithm

4.4.11. Low-level Heuristics for Bin Packing Problem

Bin Packing Problem(BPP) is different from the other three problems(Data Clustering, Graph Coloring and Exam Timetabling) studied in this research. BPP is a single objective grouping problem which it has only one objective to be optimized. The number of bins is the only objective of BPP. The single objective nature of the problem leads a little differentiation in the low-level heuristics used in the grouping hyper-heuristic. The most significant difference is there is only one best solution in the single objective optimization, therefore there is no list of non-dominating candidate solutions in the BPP problem. As stated in the low-level heuristic definitions, the first step is always selecting a candidate solution, however in BPP this step is ignored because there is only one candidate solution. The other significant difference is the repair procedure in the low-level heuristics. This repair procedure is added to all low-level heuristics defined. The reason of using this repair procedure is not to allow infeasible solutions as a candidate solution. A feasible candidate solution is formulated as $\forall_i w_i \leq C$ where w_i is the weight of the i^{th} bin and C is the maximum bin capacity. Therefore, an infeasible candidate solution contains at least one bin that exceeds the bin capacity. The repair mechanism can be seen in algorithm 4.31

```

1: for groupIndex  $\leftarrow$  0 to candidateSolution.numberOfGroups do
2:   if group.numberOfNodes == 0 then
3:     Remove group
4:   end if
5: end for
6: for groupIndex  $\leftarrow$  0 to candidateSolution.numberOfGroups do
7:   if group.weight  $\geq$  binCapacity then
8:     while group.weight  $\geq$  binCapacity do
9:       Take out the most suitable item( $n_i$ ) out
10:      Place  $n_i$  to the most suitable bin
11:      if There is no bin to put  $n_i$  then
12:        Open a new bin
13:        Place  $n_i$  to newly opened bin
14:      end if
15:    end while
16:  end if
17: end for

```

Figure 4.31. Repair Algorithm for infeasible candidate solutions

There are two points in the algorithm needs to be clarified, these are the most suitable item n_i to take out from the infeasible bin and the most suitable bin B_j to place n_i . The suitability issue in this two cases are about the fill ratios of the bins. n_i is selected to be the smallest item that makes the bin feasible and B_j is selected to be the maximum weight when n_i is placed into. This repair avoids the infeasible solutions, therefore the hyper-heuristic framework always deals with feasible solutions.

4.5. TYPES OF MULTI-OBJECTIVE HYPER-HEURISTIC FRAMEWORKS

Combination of heuristic selection and move acceptance mechanisms represents different hyper-heuristic frameworks, in addition to this hyper-heuristic frameworks can

differ from each other with different properties. In [37] proposed four different types of hyper-heuristics and these differ from each other mainly by the heuristic usages. For example, in one of the proposed hyper-heuristics, a selected hill-climber heuristic is applied to the candidate solution whenever a heuristic is applied to the candidate solution. In this research, three different types of hyper-heuristic frameworks are represented and they differ from each other in terms of candidate solution selection. These hyper-heuristics are namely; Generic hyper-heuristic(Section 4.5.1), Cyclic Candidate Solution Selection hyper-heuristic(Section 4.5.2) and Apply to All Candidate Solutions hyper-heuristic(Section 4.5.3). In multi-objective hyper-heuristic framework, after a low-level heuristic is selected a candidate solution that is represented by LLE is selected to apply a heuristic. The difference between these three hyper-heuristic frameworks is this candidate solution selection.

4.5.1. Generic Hyper-heuristic

Generic hyper-heuristic selects the candidate solution from the list of non-dominating candidate solutions completely randomly. It selects a random candidate solution and then checks if this candidate solution is suitable to apply the heuristic. As stated before, all heuristics can not be applied to all candidate solutions. Merge type heuristics and divide type heuristics have constraints. After an acceptable candidate solution is selected, heuristic is applied on it and the new fitness value of the candidate solution is calculated. If this new candidate solution is accepted by the move acceptance criteria, the candidate solution in the list of non-dominating candidate solutions is changed with the new one. The drawback of the generic hyper-heuristic is lack of equality. Some of the candidate solutions in the list of non-dominating candidate solutions may not be selected as much as the others and this may prevent them to be optimized. An iteration of a generic hyper-heuristic can be seen in Figure 4.32

4.5.2. Cyclic Candidate Solution Selection Hyper-heuristic

Cyclic Candidate Solution Selection Hyper-heuristic(CCSSH) runs completely the same with generic hyper-heuristic except its candidate solution selection. CCSSH selects a candidate solution from the list of non-dominating candidate solutions in a cyclic manner, it

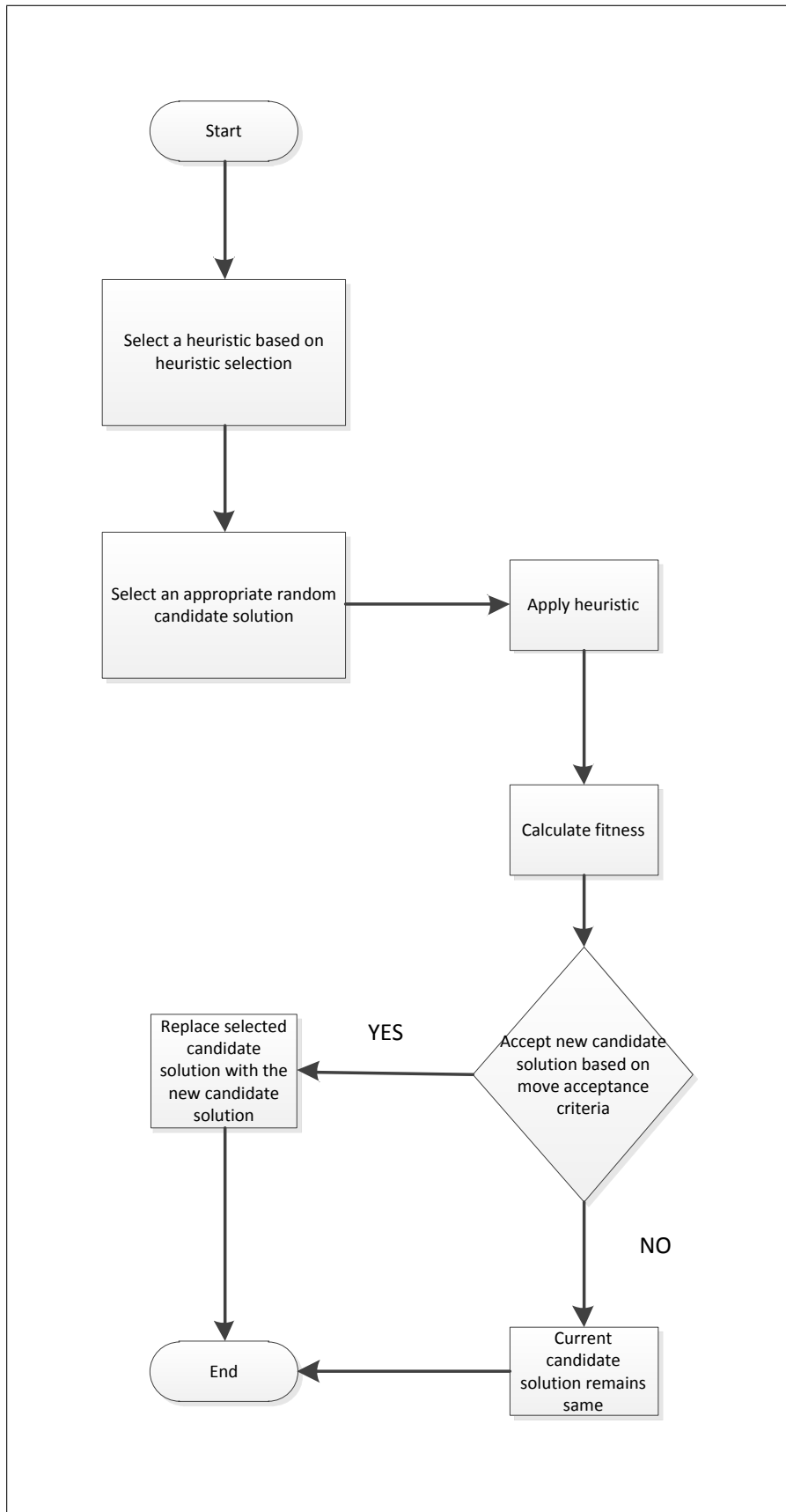


Figure 4.32. Generic Hyper-heuristic Flow Chart

selects all candidate solutions with their order in the non-dominating candidate solutions list. The aim of this hyper-heuristic is to give all points in the non-dominating candidate solutions list to optimize themselves and therefore solve the inequality problem of the generic hyper-heuristic. An iteration of CCSSH can be seen in Figure 4.33

4.5.3. Apply to All Candidate Solutions Hyper-heuristic

Apply to All Candidate Solutions Hyper-heuristic(ATACSH) also runs completely the same with generic hyper-heuristic except its candidate solution selection. ATACSH applies selected heuristic to all candidate solutions that are in the list of non-dominating candidate solutions. After a heuristic is selected by the heuristic selection mechanism, all candidate solutions in the non-dominating candidate solutions list are selected one by one with their order in the list by considering candidate solution selection constraints. This approach tries to eliminate the inequality problem in the generic hyper-heuristic. An iteration of ATACSH can be seen in Figure 4.34

4.6. SINGLE OBJECTIVE HYPER-HEURISTICS

Single objective problems are different from multi-objective problems in their nature. The difference of these two problems are stated before. The most significant difference between single and multi-objective problems is the number of solutions maintained during the search process using the grouping selection hyper-heuristic framework. Hence, while single objective problems maintain one best solution, multi-objective problems requires maintenance of a set of non-dominant solutions.

Single objective hyper-heuristic starts with an initial candidate solution and tries to improve this single solution. A heuristic is selected based on the heuristic selection mechanism and selected heuristic is applied on the candidate solution. After the heuristic is applied, a repair mechanism works on the candidate solution to repair the infeasible groups. Details of this repair mechanism is described in Algorithm 4.31. Fitness value of the new candidate solution is calculated on the feasible candidate solution. Move acceptance mechanism decides whether to accept or reject the new candidate solution and if the new

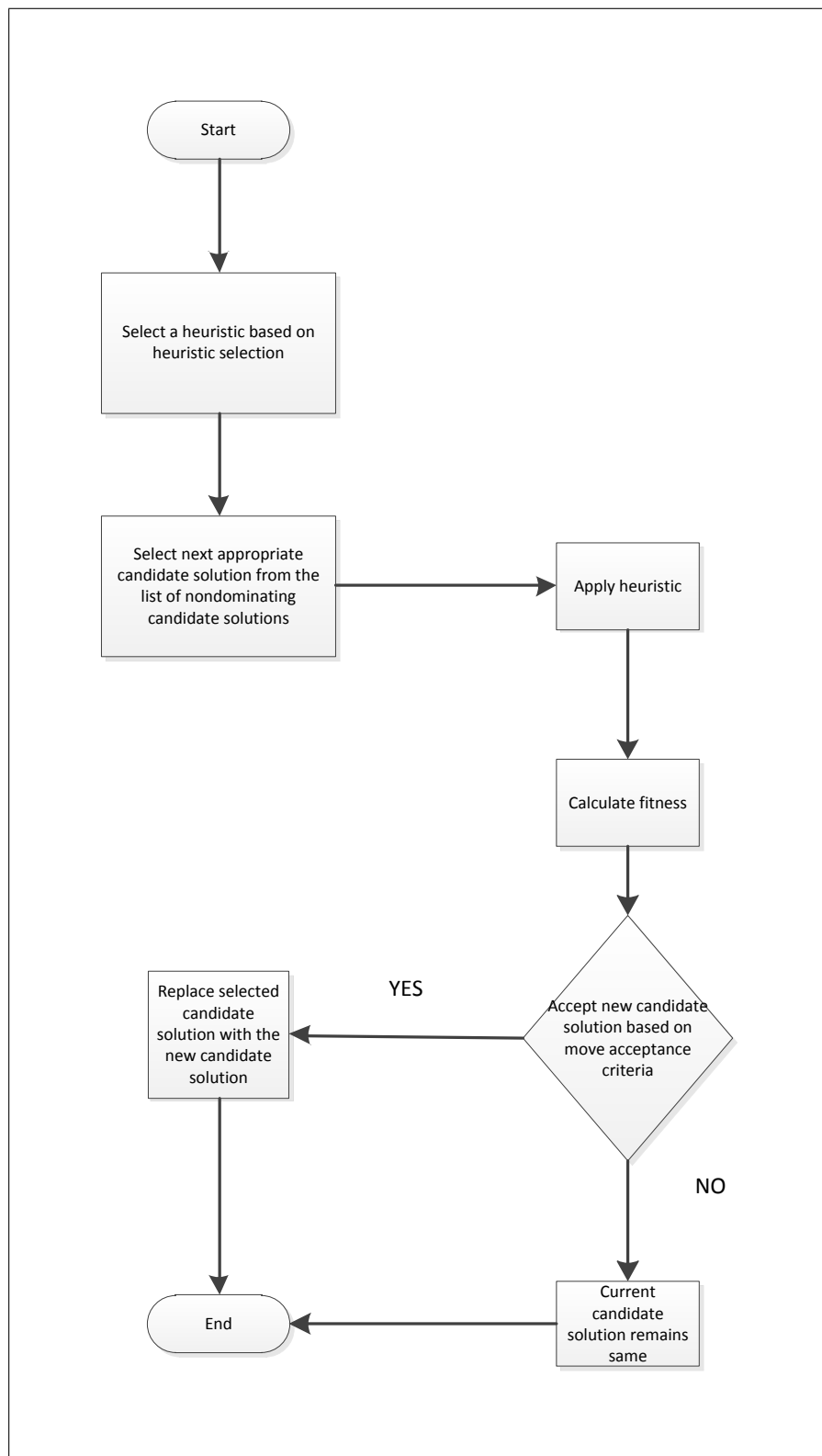


Figure 4.33. Cyclic Candidate Solution Selection Hyper-heuristic Flow Chart

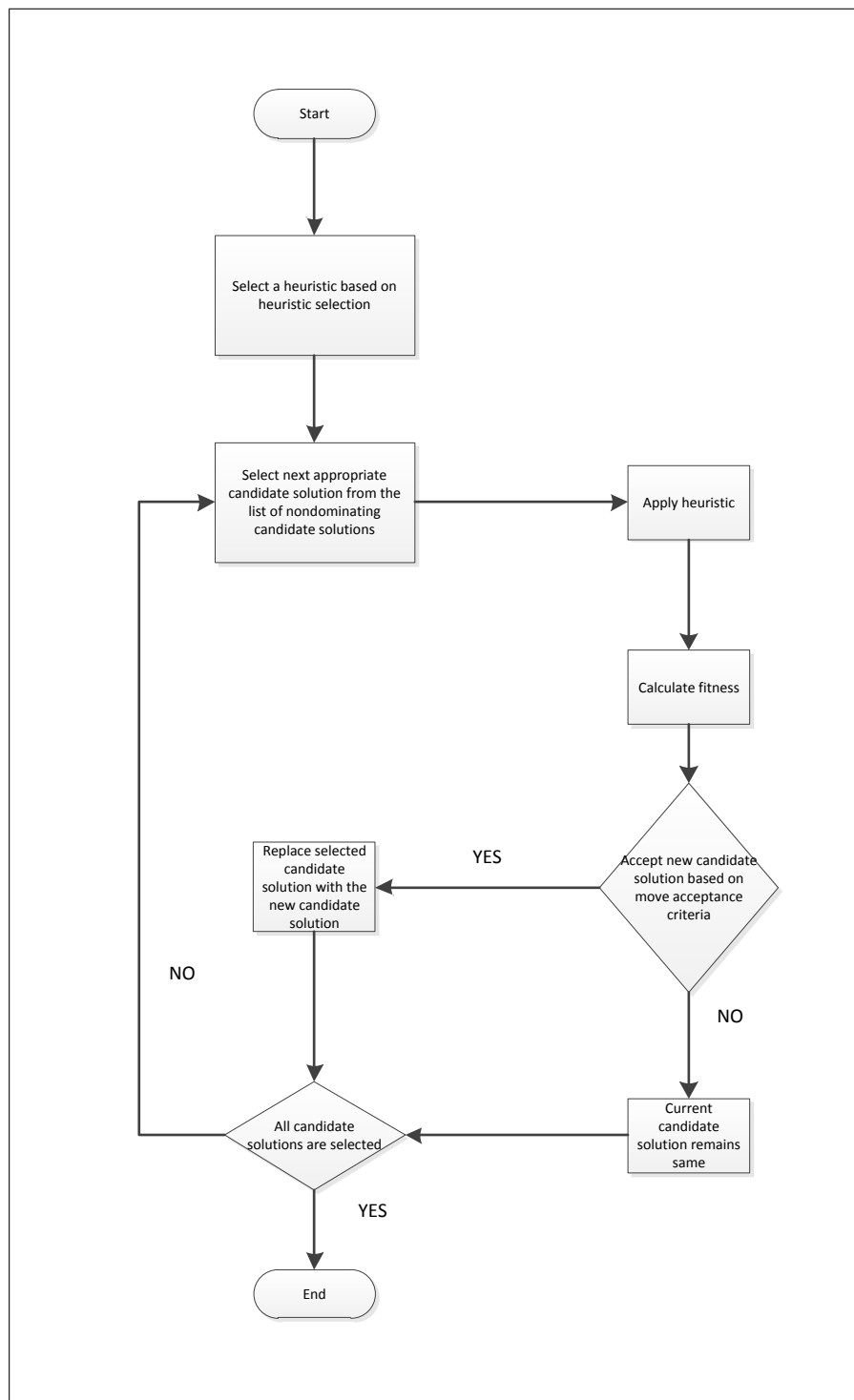


Figure 4.34. Apply to All Candidate Solutions Hyper-heuristic Flow Chart

candidate solution is accepted by the move acceptance mechanism, it is replaced with the current candidate solution. An iteration of single objective hyper-heuristic can be seen in Figure 4.35.

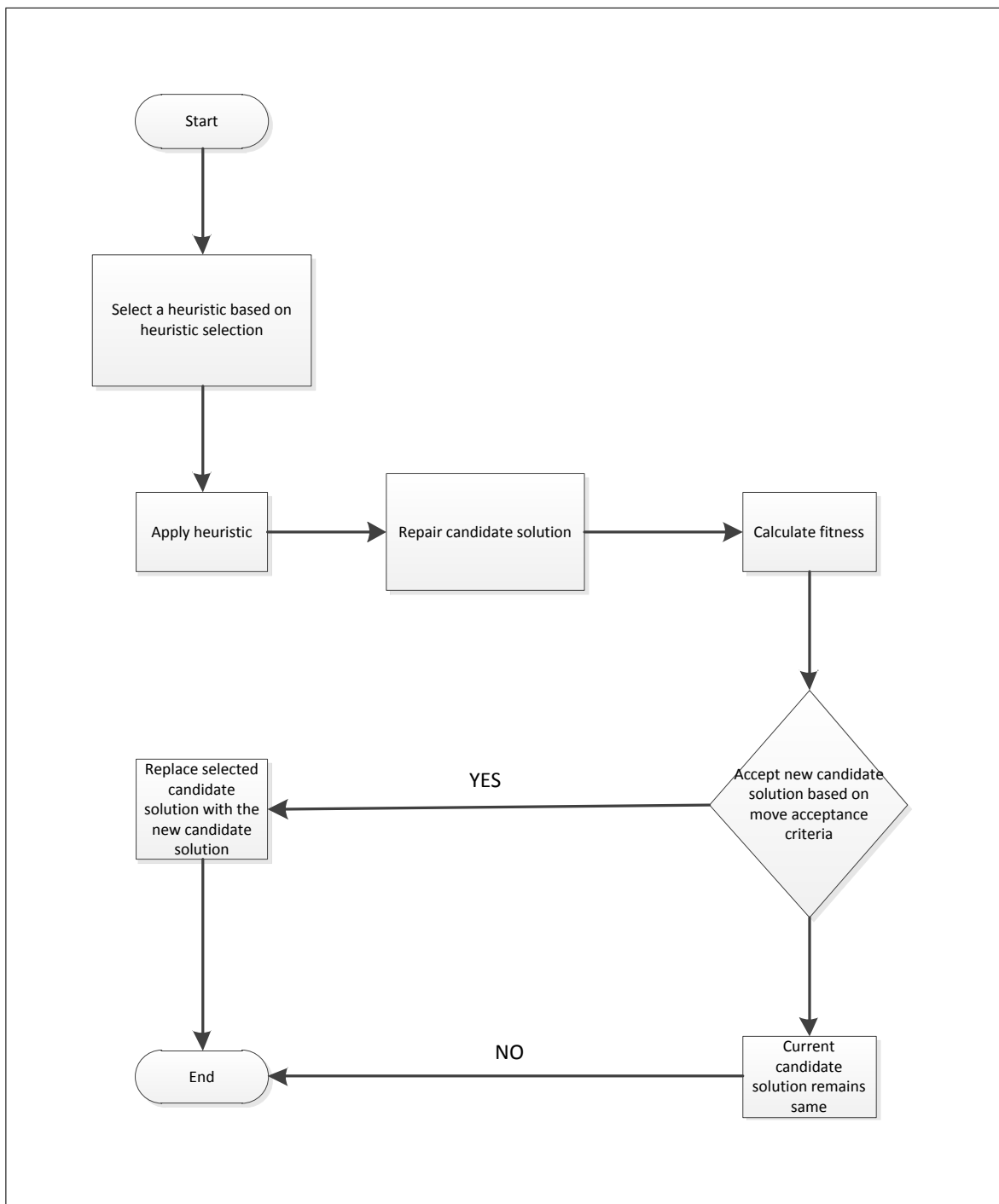


Figure 4.35. Single objective Hyper-heuristic Flow Chart

5. EXPERIMENTS

The grouping hyper-heuristic framework is tested using combinations of {Simple Random, Reinforcement Learning} heuristic selection methods and {Improving and Equal, Great Deluge, Late Acceptance} move acceptance methods. Initially, a set of parameter tuning experiments are performed. The parameter tuning experiments for the tournament size, range of pareto front, number of heuristics, hyper-heuristics types are followed by comparison of the best performing grouping hyper-heuristics and their configurations to the performance of previously proposed approaches for graph coloring, exam timetabling, data clustering and bin packing. 3 GHz quad-core Linux machines with 16 Gb memory are used during the experiments. Multiple runs are performed with each hyper-heuristic and parameter configuration for a given problem instance. All approaches are given 10 minutes of time as a termination criteria in each run.

5.1. EXPERIMENTAL DATA

The experiments for investigating the performance of grouping hyper-heuristics and different design choices are performed using a benchmark for each problem domain. DIMACS challenge suite [6] is used for graph coloring problem during the experiments. The characteristics of the instances are presented in Table 5.1. DIMACS benchmark suite is widely used by the researchers to investigate graph coloring approaches. There are different types of data instances in this benchmark suite:

- **DSJCX.Y**: Random graphs generated by Johnson et al. in [74]. The number of vertices is denoted by the first number and the second digit references the conflict density.
- **flatX_K**: These data sets are generated by partitioning the vertex set into N classes which are in equal size as much as possible and then by selecting edges only between vertices of different classes. X is the vertex set size and K is the chromatic number.
- **le450_K**: These are represented in [65] which are called Leighton graphs. 450 is the number of vertices and K is the known chromatic number.

Table 5.1. The characteristics of the problem instances from the DIMACS suite. $|V|$ is the number of vertices, $|E|$ is the number of edges, % is the edge density and $\chi(G)$ is the chromatic number.

Instance	$ V $	$ E $	%	$\chi(G)$
DSJC125.5	125	3891	0,50	?
DSJC125.9	125	6961	0,90	?
zeroin.1.col	211	4100	0,19	49
zeroin.2.col	211	3541	0,16	30
zeroin.3.col	206	3540	0,17	30
DSJC250.1	250	3218	0,10	?
DSJC250.5	250	15668	0,50	?
DSJC250.9	250	27897	0,90	?
flat300_20	300	21375	0,48	20
flat300_26	300	21633	0,48	26
flat300_28	300	21695	0,48	28
school1_nsh	352	14612	0,24	14
le450_15a	450	8168	0,08	15
le450_15b	450	8169	0,08	15
le450_15c	450	16680	0,17	15
le450_15d	450	16750	0,17	15
le450_25a	450	8260	0,08	25
le450_25b	450	8263	0,08	25
le450_25c	450	16680	0,17	25
le450_25d	450	16750	0,17	25
DSJC500.1	500	12458	0,10	?

Toronto benchmark [5] is used during the experiments for exam timetabling. The characteristics of the problem instances are presented in Table 5.2. The Toronto benchmark instances are 13 real-world exam timetabling problems which is introduced in [5]. These instances are collected from three Canadian high schools, five Canadian universities, one American university, one British university and one university in Saudi Arabia. In exam timetabling problem, the density of conflicting exams is defined with a *Conflict Matrix* C . In this matrix, each element $c_{ij} = 1$ if exam i conflicts with exam j , or $c_{ij} = 0$ otherwise. The ratio between the number of elements of value 1 to the total number of elements in the conflict matrix is called *Conflict Density*. The edge density in the Table 5.2 represents this conflict density. In the original data set there are two different objectives.

- to minimize the number of time slots needed for the problem
- to minimize the average cost per student

In this study, the first type of objective is aimed to be resolved by forming a feasible timetable with the least number of time slots. Qu et al. [79] differentiated between different versions of the data sets used in literature and proposed a naming convention which is also adapted in this study.

For data clustering, two different types of data sets are used during the experiments: real-world data set and synthetic data set. Table 5.3 presents the real-world data set and Table 5.4 presents the synthetic data set. The number of instances, number of attributes and number of expected clusters are provided in these tables. Breast, dermatology, iris and zoo are the well known real world data clustering instances. The synthetic problem instances are taken from the study of K ulah iođlu [80] for comparison of approaches. They are generated randomly based on Gaussian Distribution around randomly selected centers.

In bin packing experiments, two different sets of data set instances are used, these are provided by Falkenauer [7]. In the first set, the maximum bin capacity is set to 150 and each integer is randomly generated in a range between 20 and 100 with a uniform distribution. [7] reports that this distribution gives the most difficult problems according to the results of Martello and Toth's method [76]. Falkenauer generated instances of this kind of data sets

Table 5.2. The characteristics of the problem instances from the Toronto benchmark suite.

Instance	$ V $	$ E $	%
hec92 I	81	1363	0.42
sta83 I	139	1381	0.14
yor83 I	181	4691	0.29
ute92	184	1430	0.08
ear83 I	190	4793	0.27
tre92	261	6131	0.18
lse91	381	4531	0.06
kfu93	461	5893	0.06
rye93	486	8872	0.08
car92 I	543	20305	0.14
uta92 I	622	24249	0.13
car91 I	682	29814	0.13

Table 5.3. The characteristics of the real-world data clustering problem instances.

Instance	# of instances	# of attributes	# of clusters
Breast-cancer	699	9	2
Dermatology	366	34	6
Zoo	101	16	7
Iris	150	4	3

Table 5.4. The characteristics of the synthetic data clustering problem instances.

Instance	# of instances	# of attributes	# of clusters
Synthetic1	450	2	3
Synthetic2	500	2	5
Synthetic3	900	2	3
Synthetic4	1000	2	5

with the several number of items. These are, 120, 250, 500 and 1000 with 20 instances. In the second kind of data set, the item sizes are selected from the range (0.25, 0.50) to be packed into bins of maximum capacity 1. In these instances, a well-filled bin must contain one large item and two small items that is why Falkenauer referred them as *triplets*. Even though it is possible to pack two large items or three small items into one bin, this results an inevitable loss of space. [81] points out a similarity between triplets and 3SAT which is considered as the most difficult kSAT problem. Falkenauer finally points out that bin packing instances are easier to approximate when the number of items to be packed into a bin exceeds three, so *triplets* are the most difficult bin packing problem instances.

In order to preserve the difficulty of the problem, generated instances have pre-known local optimal point. In a maximum bin capacity of 1000, an item with a size in range [380, 490] was generated. Then left space S that is in range [510, 620], the second item's size is selected in the range $[250, S/2)$ and the third item completes the bin. Triplets of 60, 120, 249 and 501 items with 20 instances are generated. In this study, all of the instances of both kinds of data set instances, only the selected subset is used to test proposed grouping hyper-heuristic.

5.2. PARAMETER TUNING EXPERIMENTS

Several experiments are run for investigating different aspects of the hyper-heuristics. Following sections describe these experiments. In all sections selected results are listed in

tables and following abbreviations are used in tables:

- **avr**: Average solution quality based on number of groups.
- **std**: Standard deviation in average solution quality.
- **sr**: Success rate is the ratio of the number of runs in which a feasible solution is obtained to the total number of runs.
- **bst**: Best solution quality based on number of groups.

5.2.1. Tuning Tournament Size

The tournament size parameter is used in two heuristics: divide tournament and merge tournament. 2 different tournament sizes are tested, 2 and 8. A subset of data sets are used for these experiments and some of them are listed in Table 5.5. The experimental settings are as follows:

- Number of runs: 100
- Number of heuristics: 5
- Pareto front interval: 10

The experimental results show that, tour sizes 2 and 8 have the same success of finding the best solution. Both of them failed to find the expected solutions in only one instance. On average, tournament size 8 is slightly better than the tournament size 2 considering the best solutions, however tour size 2 generates a better standard deviation compared to tournament size 8 which disposes the advantage of tour size 8 on average. When we compare the success rates of the tour sizes we observe that tournament size 2 has a better performance in finding acceptable solutions. To sum up, finding the best solution neither of them is superior than the other one, however in the robustness, tour size 2 is slightly better than the tour size 8.

5.2.2. Pareto Front Interval for Multi-objective Problems

In this study, pareto front range is defined as a parameter, therefore the number of candidate solutions are known when experiments are performed. Two different pareto front

Table 5.5. Tournament Size Comparison

tour size		2				8			
instance	hyper-heuristic	avr	std	sr	bst	avr	std	sr	bst
car92	RL-GDEL	34	2,34	94	28	35	2,47	95	28
	RL-IEQ	34	2,81	87	28	34	2,79	95	28
	RL-LACC	34	2,41	71	28	33	2,65	88	28
	SRAN-GDEL	35	2,12	92	29	34	2,54	89	28
	SRAN-IEQ	35	2,30	91	28	35	2,08	91	29
	SRAN-LACC	33	2,80	78	28	33	2,74	80	28
DSJC125.9	RL-GDEL	49	1,68	89	44	49	2,22	86	44
	RL-IEQ	49	2,23	91	44	49	2,28	82	44
	RL-LACC	48	3,20	37	44	47	2,52	38	44
	SRAN-GDEL	49	2,04	90	44	49	2,27	84	44
	SRAN-IEQ	49	2,34	88	44	49	2,23	82	44
	SRAN-LACC	47	2,94	38	44	48	2,72	50	44
DSJC250.5	RL-GDEL	36	2,56	93	30	36	2,67	91	30
	RL-IEQ	36	2,57	86	30	36	2,69	90	30
	RL-LACC	35	2,57	74	30	34	2,94	70	30
	SRAN-GDEL	36	2,37	88	30	36	2,45	86	30
	SRAN-IEQ	37	2,33	92	30	36	2,96	90	30
	SRAN-LACC	34	2,88	63	30	34	2,80	60	30
yor83	RL-GDEL	26	2,37	80	19	24	2,83	78	19
	RL-IEQ	25	2,33	95	19	25	2,42	86	19
	RL-LACC	23	2,65	82	19	23	2,56	84	19
	SRAN-GDEL	25	2,6	73	19	25	2,86	77	19
	SRAN-IEQ	25	2,4	90	19	26	2,63	83	19
	SRAN-LACC	24	2,6	83	19	23	2,79	78	19

intervals are tested in the study, 5 and 10. The expected solutions are assumed to be known for all problem instances, so the candidate solutions in the pareto front are set accordingly. There is a pareto range check when applying heuristics to assure that a candidate solution is generated within the pareto range. Table 5.6 shows the results for both pareto intervals. The experimental settings are as follows:

- Number of runs: 100
- Number of heuristics: 5
- Tournament size: 2

The experimental results show that both pareto front intervals have no effect on finding the best solution. All hyper-heuristics tested in this experiment succeed to find the expected best solution. Considering the average quality and the standard deviation of these solutions, pareto front interval 5 is better in all cases. On the other hand pareto interval 10, has better results than pareto interval 5 in success rate. More candidate solutions in the pareto front generates a better success rate, however these solutions are worse and deviate with high ratios. To sum up, pareto intervals have the same best results, in the robustness comparison metric pareto interval 5 is slightly better than pareto interval 10 with its better results on average solution and standard deviation in spite of the success rate of pareto interval 10.

5.2.3. Number of Low Level Heuristics

In Section 4.4, 10 different low-level heuristics are introduced. In the experiments two different sets of low-level heuristics are used, all of them and a subset of heuristics which includes swap, divide, divide tournament, merge and merge tournament. The experimental results are summarized in Table 5.7 with the following experimental settings:

- Number of runs: 100
- Pareto Interval: 5
- Tournament size: 2

Table 5.6. Pareto Front Interval Comparison

pareto interval		10				5			
instance	hyper-heuristic	avr	std	sr	bst	avr	std	sr	bst
car92	RL-GDEL	34	2,34	94	28	31	1,3	92	28
	RL-IEQ	34	2,81	87	28	31	1,4	80	28
	RL-LACC	34	2,41	71	28	30	1,4	77	28
	SRAN-GDEL	35	2,12	92	29	31	1,4	83	28
	SRAN-IEQ	35	2,30	91	28	30	1,3	84	28
	SRAN-LACC	33	2,80	78	28	30	1,4	72	28
DSJC125.9	RL-GDEL	49	1,68	89	44	47	1,0	84	44
	RL-IEQ	49	2,23	91	44	46	1,1	86	44
	RL-LACC	48	3,20	37	44	46	1,3	48	44
	SRAN-GDEL	49	2,04	90	44	46	1,2	88	44
	SRAN-IEQ	49	2,34	88	44	46	1,0	78	44
	SRAN-LACC	47	2,94	38	44	45	1,2	32	44
DSJC250.5	RL-GDEL	36	2,56	93	30	33	1,4	78	30
	RL-IEQ	36	2,57	86	30	33	1,4	79	30
	RL-LACC	35	2,57	74	30	32	1,4	66	30
	SRAN-GDEL	36	2,37	88	30	32	1,5	81	30
	SRAN-IEQ	37	2,33	92	30	32	1,4	80	30
	SRAN-LACC	34	2,88	63	30	32	1,4	59	30
yor83	RL-GDEL	26	2,37	80	19	21	1,5	66	19
	RL-IEQ	25	2,33	95	19	22	1,3	88	19
	RL-LACC	23	2,65	82	19	21	1,3	74	19
	SRAN-GDEL	25	2,6	73	19	21	1,4	75	19
	SRAN-IEQ	25	2,4	90	19	22	1,3	86	19
	SRAN-LACC	24	2,6	83	19	21	1,4	70	19

Table 5.7. Number of Heuristics Comparison

number of heuristics		5				10			
instance	hyper-heuristic	avr	std	sr	bst	avr	std	sr	bst
car92	RL-GDEL	31	1,3	92	28	31	1,3	75	28
	RL-IEQ	31	1,4	80	28	30	1,4	79	28
	RL-LACC	30	1,4	77	28	30	1,5	74	28
	SRAN-GDEL	31	1,4	83	28	31	1,3	73	28
	SRAN-IEQ	30	1,3	84	28	30	1,4	82	28
	SRAN-LACC	30	1,4	72	28	30	1,4	71	28
DSJC125.9	RL-GDEL	47	1,0	84	44	46	1,3	63	44
	RL-IEQ	46	1,1	86	44	46	1,5	56	44
	RL-LACC	46	1,3	48	44	45	1,4	51	44
	SRAN-GDEL	46	1,2	88	44	46	1,3	72	44
	SRAN-IEQ	46	1,0	78	44	46	1,3	67	44
	SRAN-LACC	45	1,2	32	44	46	1,4	57	44
DSJC250.5	RL-GDEL	33	1,4	78	30	33	1,4	75	30
	RL-IEQ	33	1,4	79	30	32	1,4	72	30
	RL-LACC	32	1,4	66	30	32	1,4	59	30
	SRAN-GDEL	32	1,5	81	30	33	1,4	76	30
	SRAN-IEQ	32	1,4	80	30	32	1,3	60	30
	SRAN-LACC	32	1,4	59	30	32	1,5	62	30
yor83	RL-GDEL	21	1,5	66	19	21	1,5	73	19
	RL-IEQ	22	1,3	88	19	21	1,3	80	19
	RL-LACC	21	1,3	74	19	21	1,4	72	19
	SRAN-GDEL	21	1,4	75	19	21	1,4	73	19
	SRAN-IEQ	22	1,3	86	19	21	1,4	84	19
	SRAN-LACC	21	1,4	70	19	21	1,4	65	19

The experiments show that the number of low-level heuristics does not have a particular effect on the best solutions found. The average quality and the standard deviation of these solutions are very similar to each other. However, considering the success rate, hyper-heuristics with 5 low-level heuristics perform strictly better than the hyper-heuristics with 10 low-level heuristics. This might be because in the 10 low-level heuristics case, the extra heuristics are more time consuming than the other heuristics and heuristic selection methods do not take this feature into account. In particular, finding the most suitable and finding most conflicting groups take more time as compared to the other low level heuristics. To sum up, both of them is very successful for finding the best solution considering the expected best solution. However, considering the robustness measure, 5 low-level heuristics has a better performance than 10 low-level heuristics within the grouping hyper-heuristic framework.

5.2.4. Comparison of RL and RLM Heuristic Selection Methods

Modified Reinforcement Learning (RLM) is a variant of Reinforcement Learning (RL) as stated before. The scoring mechanism is modified and the scores of the low-level heuristics are updated not only if they improve the candidate solution but also when they are accepted by the move acceptance criteria. In this set of experiments, the performance of these two heuristic selection mechanisms is compared. The experimental results are shown in Table 5.8 with the following experimental settings:

- Number of runs: 100
- Pareto Interval: 10
- Tournament size: 2

In the experiments, RL and RLM generated similar and successful results in finding the best solutions for the problem instances except one. RL-IEQ failed to find the expected best solution for DSJC125.5 and RLM-IEQ failed to find for DSJC125.9. On average, two heuristic selection mechanisms generated similar results and none of them outperformed the other. When we compare the success rates of these two heuristic selection methods RL has better performance than RLM and finds more feasible solutions than RLM. The results show that the modified scoring mechanism does not improve the performance of reinforcement

Table 5.8. The performance comparison of RL and RLM heuristic selection methods.

heuristic selection		RL				RLM			
instance	move acceptance	avr	std	sr	bst	avr	std	sr	bst
car92	GDEL	34	2,3	94	28	34	2,5	94	28
	IEQ	34	2,8	87	28	33	2,8	55	28
	LACC	34	2,4	71	28	34	2,5	83	28
car91	GDEL	34	2,4	90	28	35	2,4	92	28
	IEQ	34	2,7	89	28	33	3,0	51	28
	LACC	34	2,8	77	28	34	2,6	75	28
DSJC125.5	GDEL	23	2,5	72	18	24	2,7	82	18
	IEQ	25	0,9	100	22	25	1,9	93	18
	LACC	22	2,5	79	18	22	2,8	81	18
DSJC125.9	GDEL	49	1,7	89	44	49	2,1	90	44
	IEQ	49	2,2	91	44	49	2,0	86	45
	LACC	48	3,2	37	44	47	2,6	32	44
DSJC250.5	GDEL	36	2,6	93	30	37	2,4	87	30
	IEQ	36	2,6	86	30	35	3,0	69	30
	LACC	35	2,6	74	30	35	2,8	72	30
synthetic1	GDEL	3	0,0	100	3	3	0,5	100	3
	IEQ	3	0,1	100	3	4	0,5	100	3
	LACC	3	0,0	100	3	3	0,5	100	3
synthetic2	GDEL	5	0,0	100	5	5	0,0	100	5
	IEQ	5	0,1	100	5	6	0,4	100	5
	LACC	5	0,0	100	5	6	0,5	100	5
yor83	GDEL	26	2,4	80	19	25	2,9	75	19
	IEQ	25	2,3	95	19	25	2,5	79	19
	LACC	23	2,7	82	19	23	2,6	86	19

learning heuristic selection method. To sum up, these two heuristic selection methods have similar performances based on best solution and RL has slightly better than RLM according to robustness comparison measure.

5.2.5. Comparison of Hyper-heuristic Frameworks

There are three types of hyper-heuristic frameworks proposed for multi-objective optimization in this study, namely; generic, cyclic and apply to all. These frameworks are explained in Section 4.5. They are tested with a subset of problem instances and the results are presented below.

The experimental results over exam timetabling problem instances considering average and best performances are summarized in Table 5.9 and Table 5.10. In Table 5.10, it can be observed that all hyper-heuristic types can reach the expected solution quality. The only exception to this is the cyclic type hyper-heuristic which failed to find the expected solution quality for yor83 when the SRAN–IEQ hyper-heuristic is used. The average performance of hyper-heuristic types for exam timetabling problems are similar to each other. While cyclic hyper-heuristic type’s success rate is slightly better than the others, generic hyper-heuristic type’s average performance are slightly better. To sum up, all three hyper-heuristic types deliver similar average performances but cyclic and generic types are slightly better than the apply all hyper-heuristic type. Considering the best performance of runs criterion, apply to all and generic types are the same and they perform slightly better than the cyclic type hyper-heuristic.

The results of the graph coloring problem instances are presented in Table 5.11 and Table 5.12. Table 5.11 compares the average performances of the hyper-heuristic types and Table 5.12 compares the best performance of the hyper-heuristic types. All three hyper-heuristic types find the same best solutions in all experiment cases but the apply to all hyper-heuristic type failed to find the same best solution with the other two hyper-heuristic types in DSJC125.5 problem instance when run with SRAN–IEQ hyper-heuristic. On the average performance of hyper-heuristic types, they have similar average performances for finding close results to expected best solutions, when these are combined with the standard deviation

Table 5.9. Hyper-heuristic types average performance comparison on Exam Timetabling Problems

hh type		generic			cyclic			all		
instance	hh	avr	std	sr	avr	std	sr	avr	std	sr
car92	RL-GDEL	30	1,4	66	30	1,5	76	30	1,5	80
	RL-IEQ	30	1,4	70	31	1,3	72	30	1,3	90
	RL-LACC	30	1,4	84	30	1,3	82	30	1,3	80
	SRAN-GDEL	31	1,1	74	30	1,6	74	30	1,5	76
	SRAN-IEQ	30	1,4	80	31	1,3	84	31	1,3	80
	SRAN-LACC	30	1,4	76	30	1,3	72	30	1,3	70
car91	RL-GDEL	31	1,3	84	31	1,3	72	30	1,5	88
	RL-IEQ	31	1,1	78	30	1,4	72	31	1,3	84
	RL-LACC	30	1,6	70	30	1,6	80	30	1,5	78
	SRAN-GDEL	30	1,4	78	31	1,4	76	31	1,5	72
	SRAN-IEQ	31	1,3	88	30	1,5	72	30	1,6	76
	SRAN-LACC	31	1,2	76	31	1,5	80	31	1,3	72
sta83	RL-GDEL	13	0,4	100	13	0,3	100	13	0,4	100
	RL-IEQ	13	0,0	100	13	0,0	100	13	0,0	100
	RL-LACC	14	0,8	54	14	1,3	56	14	1,0	50
	SRAN-GDEL	13	0,4	100	13	0,4	100	13	0,4	100
	SRAN-IEQ	13	0,0	100	13	0,0	100	13	0,0	100
	SRAN-LACC	14	1,2	50	14	1,0	52	14	1,1	50
yor83	RL-GDEL	21	1,5	70	21	1,4	82	21	1,5	70
	RL-IEQ	22	1,1	98	22	1,1	96	22	1,2	96
	RL-LACC	20	1,4	66	21	1,5	60	21	1,6	58
	SRAN-GDEL	21	1,5	70	21	1,4	72	22	1,3	80
	SRAN-IEQ	22	0,8	94	22	0,7	96	22	1,0	88
	SRAN-LACC	21	1,6	64	21	1,1	70	21	1,4	62

Table 5.10. Hyper-heuristic types best performance comparison on Exam Timetabling Problems

hh type		generic	cyclic	all
instance	hh	best	best	best
car92	RL-GDEL	28	28	28
	RL-IEQ	28	28	28
	RL-LACC	28	28	28
	SRAN-GDEL	28	28	28
	SRAN-IEQ	28	28	28
	SRAN-LACC	28	28	28
car91	RL-GDEL	28	28	28
	RL-IEQ	28	28	28
	RL-LACC	28	28	28
	SRAN-GDEL	28	28	28
	SRAN-IEQ	28	28	28
	SRAN-LACC	28	28	28
sta83	RL-GDEL	13	13	13
	RL-IEQ	13	13	13
	RL-LACC	13	13	13
	SRAN-GDEL	13	13	13
	SRAN-IEQ	13	13	13
	SRAN-LACC	13	13	13
yor83	RL-GDEL	19	19	19
	RL-IEQ	19	19	19
	RL-LACC	19	19	19
	SRAN-GDEL	19	19	19
	SRAN-IEQ	19	20	19
	SRAN-LACC	19	19	19

of these results, generic hyper-heuristic type is slightly better than the other two types. In the success rate comparison metric, apply to all hyper-heuristic type has a very slightly better performance than the other two hyper-heuristic types. To sum up, these three hyper-heuristic types have similar average and best performances again, but in best case generic and cyclic types are slightly better than the apply to all type and on average case they are not separated significantly with the results.

The results of the data clustering problem instances are presented in Table 5.13 and Table 5.14. Table 5.13 compares the average performances of the hyper-heuristic types on data clustering problem instances and Table 5.14 compares the best performance of the hyper-heuristic types on data clustering problem instances. Generic hyper-heuristic type failed to find acceptable solution in RL-IEQ and SRAN-IEQ hyper-heuristics in dermatology problem instance and apply to all hyper-heuristic failed to find acceptable solutions in RL-IEQ, SRAN-GDEL and SRAN-IEQ hyper-heuristics. Cyclic hyper-heuristic type succeed to find acceptable solutions in every hyper-heuristics. However they all failed to find the expected best quality solution for dermatology problem instance. In zoo problem instance, all of the hyper-heuristic types are performed the same and they all failed to find the expected best quality solution. On the average performance of these three hyper-heuristic types, all of them performed the same in zoo problem instance and cyclic type is slightly better than generic type and generic type is slightly better than apply to all type. To sum up, when we combine the best and average performances, cyclic hyper-heuristic type performed better than the other two hyper-heuristic types, however when they are compared to state-of-art solutions, they all performed bad.

5.3. EXPERIMENTAL RESULTS FOR THE MULTI-OBJECTIVE PROBLEMS

In Section 5.2, we investigated several parameters of hyper-heuristic frameworks. In this section, results of the selected configuration are presented and they are compared with older studies. After parameter tuning experiments, the *generic hyper-heuristic type* performed slightly better than the other two types, so it is selected for this set of experiments. Also tour size 2, pareto interval 5 and number of heuristics 5 performed better than their counterparts in the experiments. The experiments show that reinforcement learning modified

Table 5.11. Hyper-heuristic types average performance comparison on Graph Coloring Problems

hh type		generic			cyclic			all		
instance	hh	avr	std	sr	avr	std	sr	avr	std	sr
DSJC125.5	RL-GDEL	21	1,3	78	21	1,4	80	20	1,5	82
	RL-IEQ	20	0,6	100	20	0,5	100	20	0,6	100
	RL-LACC	19	1,3	62	19	1,5	54	20	1,3	56
	SRAN-GDEL	20	1,2	76	21	1,3	66	20	1,2	78
	SRAN-IEQ	20	0,7	100	20	0,7	100	21	0,7	100
	SRAN-LACC	20	1,2	62	20	1,3	70	20	1,4	62
DSJC125.9	RL-GDEL	46	1,2	66	46	1,2	68	46	1,4	76
	RL-IEQ	46	1,3	88	46	1,4	94	46	1,3	80
	RL-LACC	46	1,4	60	46	1,5	64	46	1,5	62
	SRAN-GDEL	46	1,3	74	46	1,2	74	47	1,3	78
	SRAN-IEQ	47	1,1	86	47	1,2	86	46	1,3	78
	SRAN-LACC	46	1,4	70	46	1,5	60	46	1,4	60
DSJC250.1	RL-GDEL	12	1,2	88	11	1,3	80	12	1,3	80
	RL-IEQ	11	0,4	100	10	0,4	100	10	0,4	100
	RL-LACC	12	1,2	70	12	1,1	64	12	1,1	82
	SRAN-GDEL	12	1,2	88	12	1,4	82	12	1,1	92
	SRAN-IEQ	11	0,5	100	10	0,5	100	11	0,5	100
	SRAN-LACC	12	1,2	80	12	1,3	66	11	1,4	80
zeroin.i.3	RL-GDEL	33	0,8	100	33	0,8	100	33	0,7	100
	RL-IEQ	30	0,4	100	30	0,5	100	30	0,3	100
	RL-LACC	32	1,5	50	31	1,3	64	32	1,4	50
	SRAN-GDEL	32	0,8	100	33	0,8	100	33	0,8	100
	SRAN-IEQ	30	0,4	100	30	0,6	100	30	0,4	100
	SRAN-LACC	31	1,3	42	32	1,2	58	32	1,4	54

Table 5.12. Hyper-heuristic types best performance comparison on Graph Coloring Problems

hh type		generic	cyclic	all
instance	hh	best	best	best
DSJC125.5	RL-GDEL	18	18	18
	RL-IEQ	19	19	19
	RL-LACC	18	18	18
	SRAN-GDEL	18	18	18
	SRAN-IEQ	19	19	20
	SRAN-LACC	18	18	18
DSJC125.9	RL-GDEL	44	44	44
	RL-IEQ	44	44	44
	RL-LACC	44	44	44
	SRAN-GDEL	44	44	44
	SRAN-IEQ	44	44	44
	SRAN-LACC	44	44	44
DSJC250.1	RL-GDEL	9	9	9
	RL-IEQ	10	10	10
	RL-LACC	9	9	9
	SRAN-GDEL	9	9	9
	SRAN-IEQ	10	10	10
	SRAN-LACC	9	9	9
zeroin.i.3	RL-GDEL	31	31	31
	RL-IEQ	30	30	30
	RL-LACC	30	30	30
	SRAN-GDEL	31	31	31
	SRAN-IEQ	30	30	30
	SRAN-LACC	30	30	30

Table 5.13. Hyper-heuristic types average performance comparison on Data Clustering Problems

hh type		generic			cyclic			all		
instance	hh	avr	std	sr	avr	std	sr	avr	std	sr
dermatology	RL-GDEL	6	0,5	16	6	0,0	6	6		2
	RL-IEQ			0	6		2			0
	RL-LACC	6	0,5	14	7	0,8	14	6		2
	SRAN-GDEL	7	0,6	8	7	1,0	12			0
	SRAN-IEQ			0	6		2			0
	SRAN-LACC	6	0,4	10	6	0,0	4	7	0,6	8
zoo	RL-GDEL	10	0,0	100	10	0,0	100	10	0,0	100
	RL-IEQ	10	0,0	100	10	0,0	100	10	0,0	100
	RL-LACC	10	0,0	100	10	0,0	100	10	0,0	100
	SRAN-GDEL	10	0,0	100	10	0,0	100	10	0,0	100
	SRAN-IEQ	10	0,0	100	10	0,0	100	10	0,0	100
	SRAN-LACC	10	0,0	100	10	0,0	100	10	0,0	100

Table 5.14. Hyper-heuristic types best performance comparison on Data Clustering Problems

hh type		generic	cyclic	all
instance	hh	best	best	best
dermatology	RL-GDEL	6	6	6
	RL-IEQ		6	
	RL-LACC	6	6	6
	SRAN-GDEL	6	6	
	SRAN-IEQ		6	
	SRAN-LACC	6	6	6
zoo	RL-GDEL	10	10	10
	RL-IEQ	10	10	10
	RL-LACC	10	10	10
	SRAN-GDEL	10	10	10
	SRAN-IEQ	10	10	10
	SRAN-LACC	10	10	10

has no performance gain over reinforcement learning. 50 runs are run with this configuration and all 21 graph coloring problem instances, 8 data clustering problem instances and 12 exam timetabling problem instances are tested. 6 hyper-heuristics are used to test these problem instances, these are namely; RL–GDEL, RL–IEQ, RL–LACC, SRAN–GDEL, SRAN–IEQ and SRAN–LACC.

In order to compare the performances of hyper-heuristics, a ranking mechanism based on best solutions found is used. The most successful hyper-heuristic is ranked 1 and the worst one ranked 6. The ties between hyper-heuristics are taken into account and ranked according to this. Average ranks of the hyper-heuristics are taken and the hyper-heuristics with best performances are decided by this ranking procedure. All problem domains are taken into account separately, ranking of the graph coloring problems is shown in Figure 5.1. In this figure, it can be seen that RL–IEQ, RL–LACC and SRAN–LACC performed the same and they are slightly better than the others. Rankings of the hyper-heuristics on data clustering problems is presented in Figure 5.2. In data clustering RL–GDEL and SRAN–GDEL performed slightly better than the others. However in data clustering, hyper-heuristics failed to find expected best quality solutions in three of the four real world date sets. Rankings of hyper-heuristics on exam timetabling problems is shown in Figure 5.3. RL–IEQ, RL–LACC and SRAN–LACC performed the same and they are slightly better than the other hyper-heuristics. In general, RL–LACC and SRAN–LACC performed slightly better than the other hyper-heuristics.

RL–LACC and SRAN–LACC are the two best performers of the hyper-heuristics among the six hyper-heuristics tested. The best solutions of these are compared with old studies [56, 80, 82]. RL–LACC and SRAN–LACC have the same best performances so they are denoted as (*-LACC). Greedy Partition Crossover Lowest Index (GPX-LI), Greedy Partition Crossover Cardinality Based (GPX-CB) and Lowest Index Max Crossover (LIMX) graph coloring algorithms are proposed in [56]. Kirovski B (Kir-B) and Kirovski C (Kir C) graph coloring algorithms are proposed in [82]. Linear Linkage Encoding With Ending Node Links (LLE-e) and Linear Linkage Encoding With Backward Links (LLE-b) are proposed representation schemes that are used with genetic operators in [80]. This study also, tested these representation schemes with classical LLE (LLE). The fields marked with “?” means

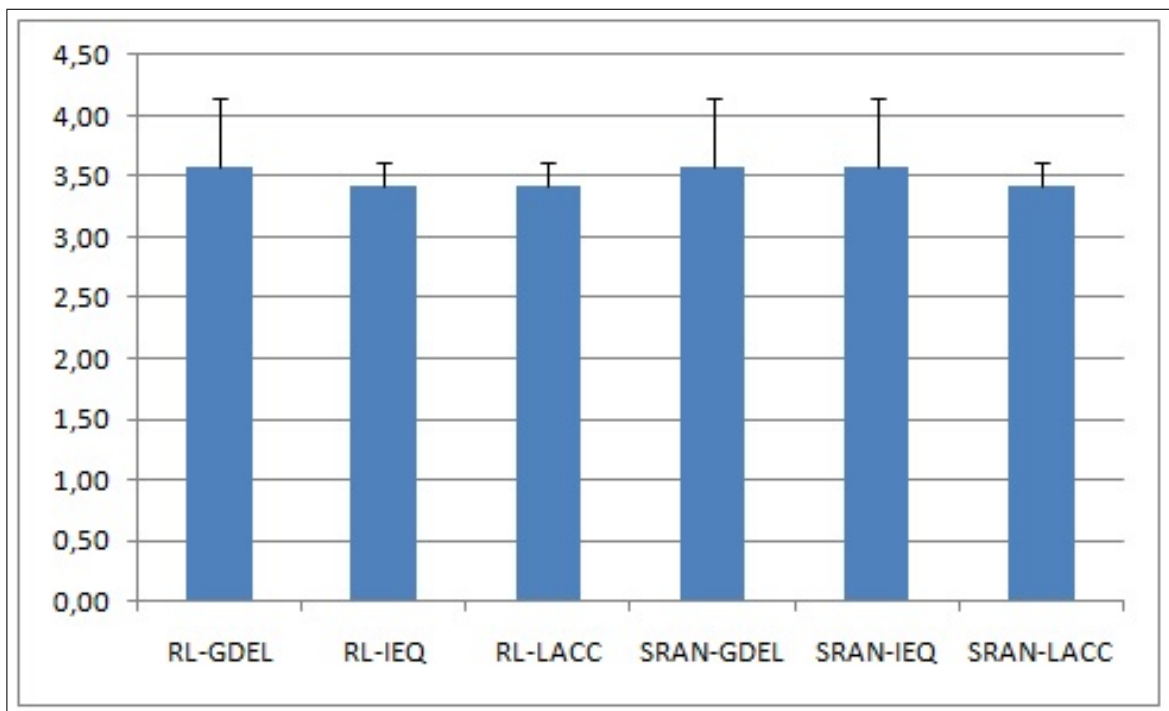


Figure 5.1. Graph Coloring Ranking based on the Best Solutions

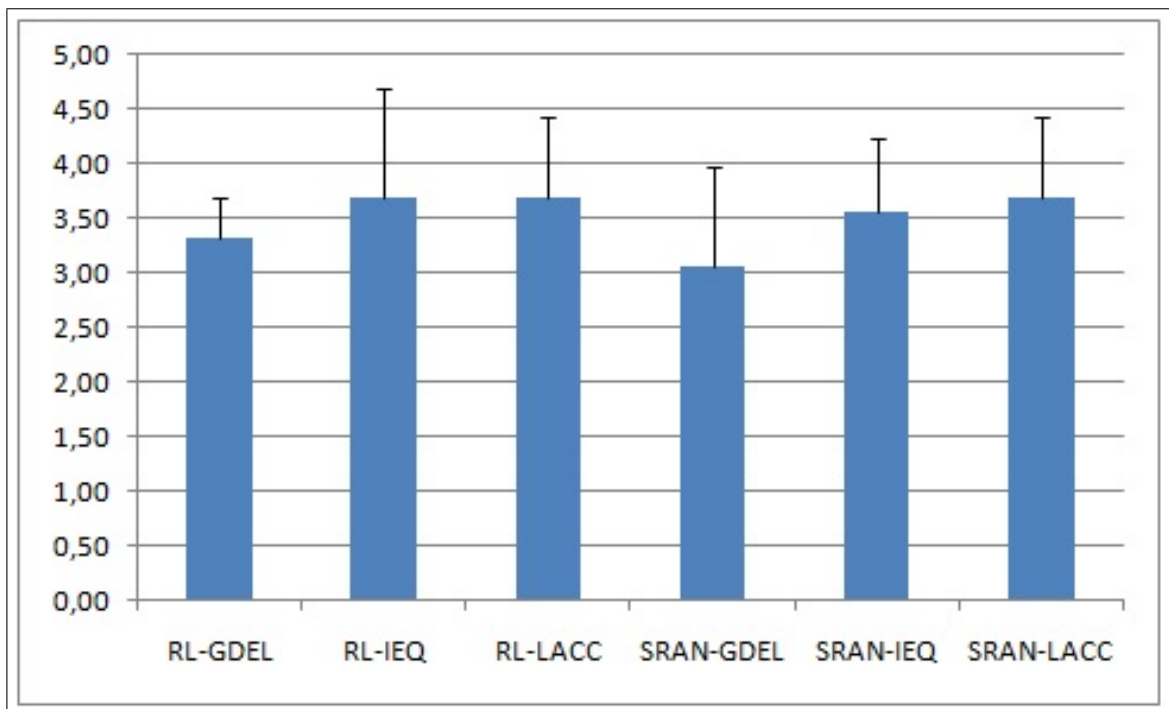


Figure 5.2. Data Clustering Ranking based on the Best Solutions

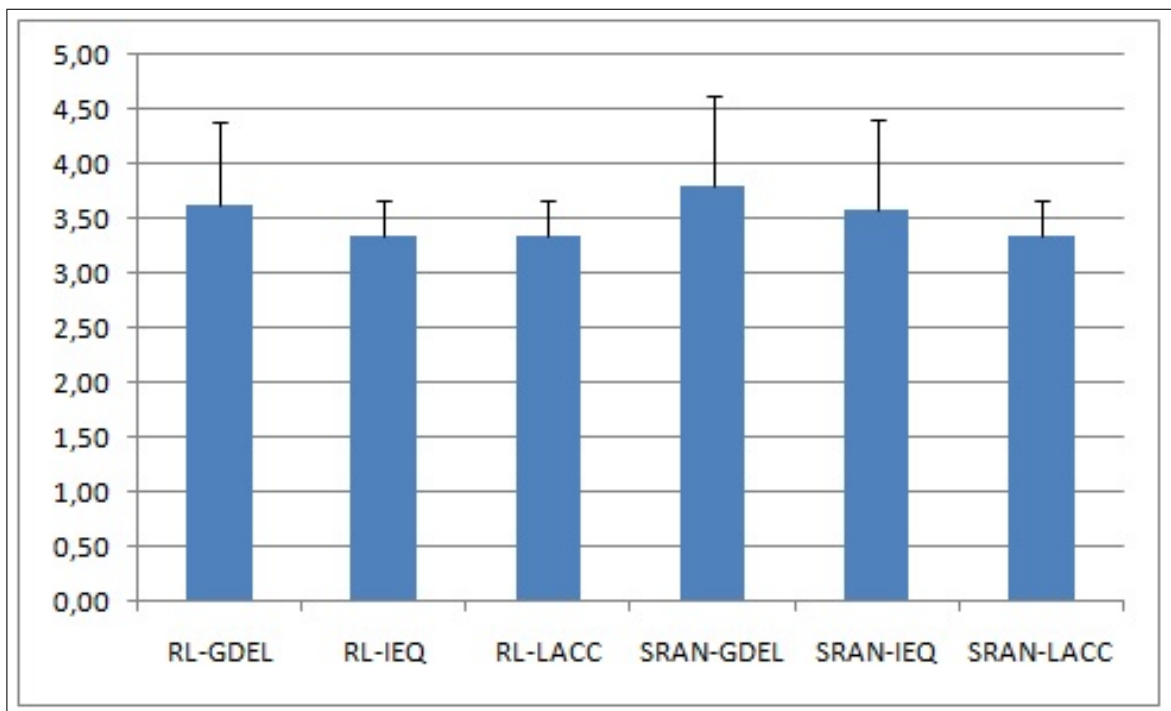


Figure 5.3. Exam Timetabling Ranking based on Best Solutions

there is no solution reported for that problem instance in that hyper-heuristic. The best colorings of these experiments are shown in Table 5.15.

The results of RL-LACC and SRAN-LACC are competitive with the previous studies and they outperform the old studies in some problem instances. The grouping hyper-heuristics proposed in this study succeeded to find the expected best quality solutions in all problem instances.

In exam timetabling data sets, RL-LACC and SRAN-LACC are compared with studies of Carter et. al. [83], Caramia et. al. [84], Merlot et. al. [85] and Ulker et. al. [56]. Greedy Partition Crossover Lowest Index (GPX-LI), Greedy Partition Crossover Cardinality Based (GPX-CB) and Lowest Index Max Crossover (LIMX) graph coloring algorithms are presented in [56]. RL-LACC and SRAN-LACC are denoted as (*-L), because their best performances are same. The results are shown in Table 5.16.

The results show that hyper-heuristics presented in this study have competitive

Table 5.15. Graph Coloring best colorings comparison

Instance	*-L	LIMX	GPX-LI	GPX-CB	LLE-e	LLE-b	LLE	Kir-B	Kir-C
DSJC125.5	18	18	18	18	19	19	18	19	18
DSJC125.9	44	44	44	44	44	44	44	45	45
zeroin.i.1	49	49	49	49	?	?	?	49	49
zeroin.i.2	30	31	31	31	?	?	?	30	30
zeroin.i.3	30	31	30	31	?	?	?	30	30
DSJC250.1	9	9	9	9	9	9	9	9	9
DSJC250.5	30	31	31	31	?	?	?	30	30
DSJC250.9	74	75	75	74	74	74	74	77	77
Flat300_20	20	20	27	32	20	32	33	20	20
Flat300_26	26	34	34	34	?	?	?	32	28
Flat300_28	28	34	34	34	?	?	?	33	32
school1_nsh	14	14	14	14	14	14	14	16	14
le450_15a	15	16	16	16	?	?	?	17	17
le450_15b	15	16	16	16	17	17	17	17	17
le450_15c	15	23	23	23	?	?	?	22	21
le450_15d	15	23	23	23	?	?	?	22	21
le450_25a	25	25	25	25	?	?	?	25	25
le450_25b	25	25	25	25	?	?	?	25	25
le450_25c	25	28	28	28	29	29	29	28	28
le450_25d	25	28	28	28	?	?	?	?	?
DSJC500.1	14	14	14	14	?	?	?	14	14

Table 5.16. Exam Timetabling best colorings comparison

Instance	*-L	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	22	23	24	23	22	22	24
tre92	20	21	21	21	20	20	21
lse91	17	17	18	18	17	17	18
kfu93	19	20	20	20	19	19	21
rye93	21	23	23	23	21	21	22
car92	28	36	36	36	28	28	31
uta92	30	38	38	38	32	30	32
car91	28	36	37	35	28	28	30

Table 5.17. Data Clustering best solutions comparison

Instance	RL-LACC	SRAN-LACC	Expected Best
breast	6	6	2
dermatology	7	7	5
iris	3	3	3
zoo	10	10	7
synthetic1	3	3	3
synthetic2	5	5	5
synthetic3	3	3	3
synthetic4	5	5	5

performances in all problem instances with all previous studies. Also they perform better in most cases to the previous studies.

In data clustering problem instances RL–LACC and SRAN–LACC are compared with the expected best solutions of the problem instances. The results are shown in Table 5.17

These are the two pareto fronts from the tests run. The iris problem instance in Figure 5.4 and synthetic2 problem instance in Figure 5.5. The expected best quality solution points can be clearly seen in these two pareto fronts. Points in the pareto fronts that represent the candidate solution with three groups in iris pareto front and the candidate solution with five groups in the synthetic2 pareto front are clearly shown as the best quality points.

Hyper-heuristics presented in this study failed to find the expected best solutions in three of the four real-world problem instances. Only in iris problem instance, hyper-heuristics succeeded to find the expected best solutions. However in the synthetic data clustering problem instances, hyper-heuristics performed well and succeeded to find the expected best solutions.

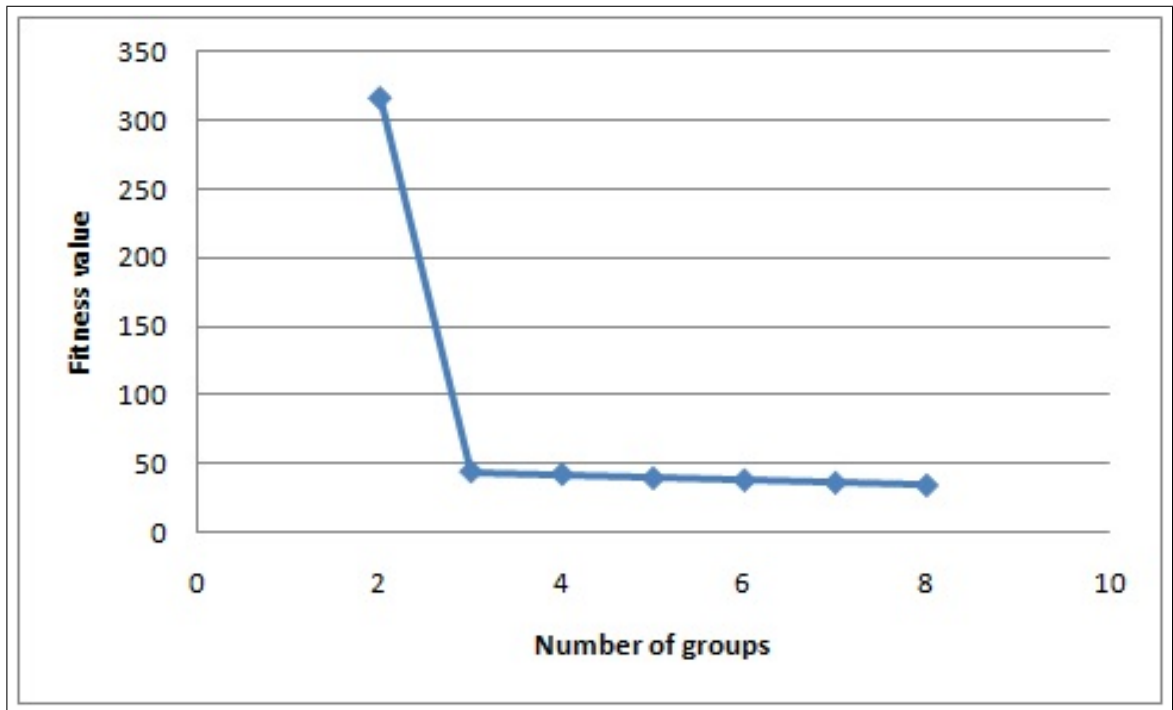


Figure 5.4. Pareto front of iris data with hyper-heuristic RL-LACC

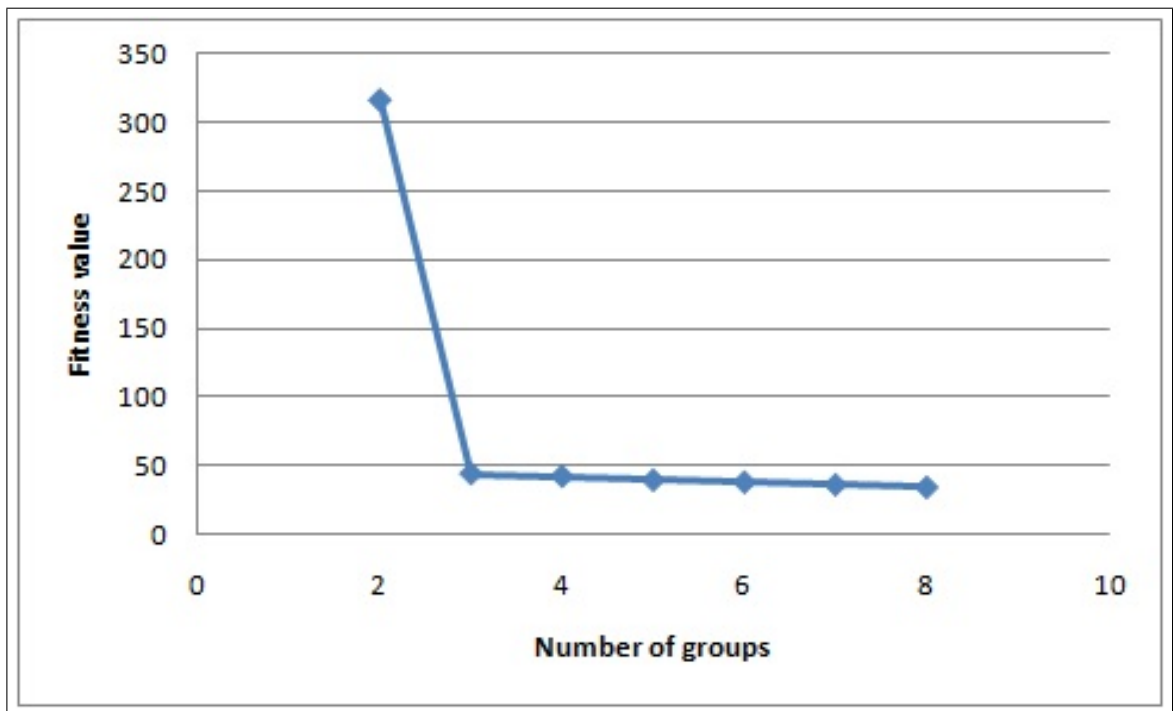


Figure 5.5. Pareto front of synthetic2 data with hyper-heuristic SRAN-LACC

Table 5.18. Bin Packing average best solutions comparison

hh	t60	t120	t249	t501	u120	u250	u500	u1000	Avrg.
RL-GDEL	22	43,67	92	184,67	50,33	106,67	209,67	420,67	141,21
RL-IEQ	22	44	91,33	184,33	50,33	106,67	209,33	420,67	141,08
RL-LACC	21	43	90	181,33	50,33	106,33	209,67	421	140,33
RLM-GDEL	22	44	90,33	184,67	50,33	106,33	209,67	421	141,04
RLM-IEQ	22	44	91	184,67	50,33	106,33	209,33	420,33	141
RLM-LACC	21	42	89	176,67	49,67	105,33	209	420,33	139,13
SRAN-GDEL	22	44	90,33	184,67	50,33	106,33	210	421,33	141,13
SRAN-IEQ	22	44	90,67	184,67	50,33	106,33	209,67	420,33	141
SRAN-LACC	21,33	43	90,33	181,67	50,33	106	210	421	140,46

5.4. EXPERIMENTAL RESULTS FOR BIN PACKING

Bin packing experiments are done with 9 different hyper-heuristics. These are RL-GDEL, RL-IEQ, RL-LACC, RLM-GDEL, RLM-IEQ, RLM-LACC, SRAN-GDEL, SRAN-IEQ and SRAN-LACC. 7 low-level heuristics are used for these hyper-heuristics, these heuristics are; swap, merge, merge tournament, divide, divide tournament, change, change to most suitable. Tournament size for merge tournament and divide tournament is set to 2. The average best solutions of each hyper-heuristic is shown in Table 5.18. According to average best solutions hyper-heuristics that use Late Acceptance move acceptance method perform better than the other hyper-heuristics and Modified Reinforcement Learning with Late Acceptance has better than the other two Late Acceptance hyper-heuristics. Modified Reinforcement Learning has performed slightly better than its original version, reinforcement learning.

The performances of hyper-heuristics are also compared with the ranking mechanism used in the multi-objective optimization experiments. Best solutions found for each problem instance is taken into account. Hyper-heuristic that finds the best solution gets the rank 1 and hyper-heuristic that finds the worst solution gets rank 9, ties are taken into account. The results are shown in Figure 5.6.

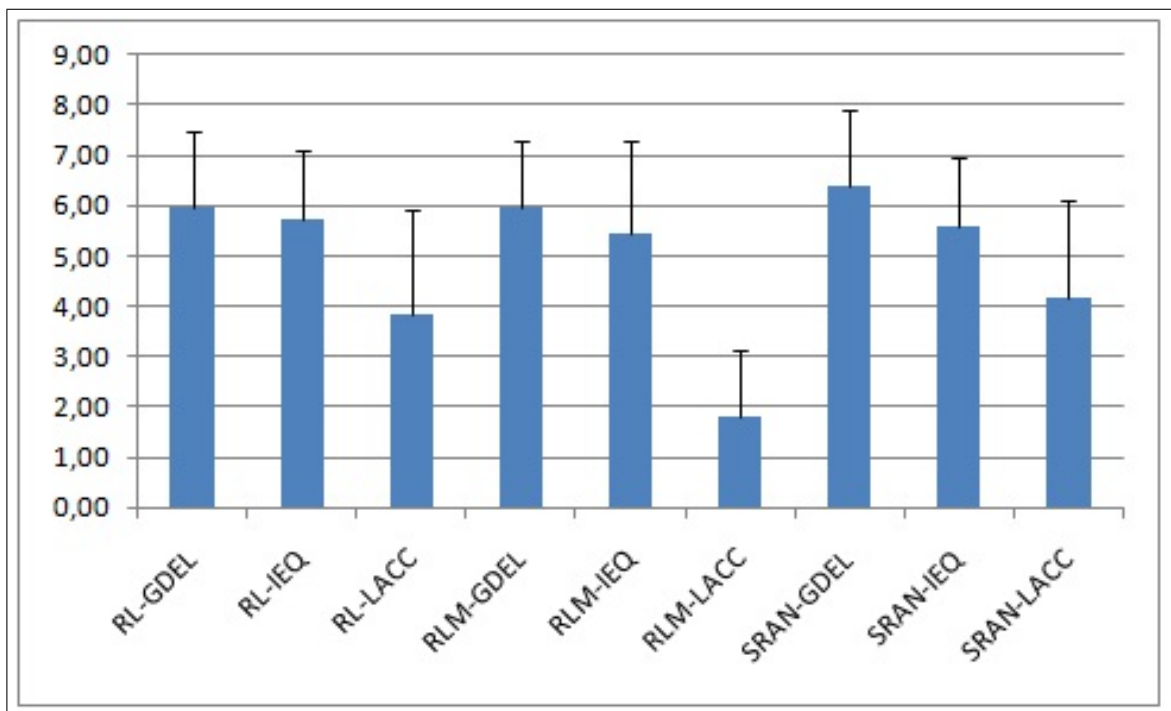


Figure 5.6. Bin Packing Problem Ranking based on Best Solutions

These results also show that, Late Acceptance move acceptance method has better performance than great deluge and improve or equal move acceptance methods. RLM–LACC has superior performance over the other eight hyper-heuristics.

In the two comparisons RLM–LACC resulted as performing better than the other hyper-heuristics. Therefore the pairwise performance variations between RLM–LACC and other hyper-heuristics are compared to observe the difference is statistically significant or not. The results are shown in Table 5.19. In this table, the heuristic selection methods are presented as R,M and S. R stands for Reinforcement Learning, M stands for Modified Reinforcement Learning and S stands for Simple Random. Move acceptance methods are presented as G for Great Deluge, I for Improve or Equal and L for Late Acceptance. Pairwise comparison using T-test of hyper-heuristics for each type of hyper-heuristics determined by a given frequency and severity of change. Given A vs B, $s+$ ($s-$) denote that A (B) is performing statistically better than B (A), while \approx denotes that there is no statistically significant performance variation between A and B.

Table 5.19. Pair-wise comparison of hyper-heuristics

Instance	RLM-LACC							
	R-G	R-I	R-L	M-G	M-I	S-G	S-I	S-L
t60_01	s+	s+	≈	s+	s+	s+	s+	≈
t60_02	s+	s+	≈	s+	s+	s+	s+	≈
t60_03	s+	s+	≈	s+	s+	s+	s+	≈
t120_01	s+	s+	≈	s+	s+	s+	s+	≈
t120_02	s+	s+	s+	s+	s+	s+	s+	s+
t120_03	s+	s+	s-	s+	s+	s+	s+	s-
t249_01	s+	s+	s-	s+	s+	s+	s+	≈
t249_02	s+	s+	≈	s+	s+	s+	s+	≈
t249_03	s+	s+	≈	s+	s+	s+	s+	≈
t501_01	s+	s+	≈	s+	s+	s+	s+	≈
t501_02	s+	s+	≈	s+	s+	s+	s+	≈
t501_05	s+	s+	≈	s+	s+	s+	s+	≈
u120_01	≈	s+	≈	s+	s+	s+	s+	≈
u120_02	s+	s+	≈	s+	s+	s+	s+	≈
u120_07	≈	s+	≈	s+	≈	s+	s+	≈
u250_04	s+	s+	s+	s+	s+	s+	s+	s+
u250_08	s+	s+	s+	s+	s+	s+	s+	s+
u250_12	s+	s+	s+	s+	s+	s+	s+	s+
u500_02	s+	s+	s+	≈	s+	s+	s+	≈
u500_08	s+	s+	s+	s+	s+	s+	s+	s+
u500_09	s+	s+	s+	s+	s+	s+	s+	s+
u1000_01	≈	≈	s+	≈	s+	s+	s+	≈
u1000_03	s+	s+	s+	s+	s+	s+	s+	s+
u1000_10	≈	≈	≈	≈	s+	s+	≈	s+

Table 5.20. Bin packing problem mean number of bins comparison

hh	t60	t120	t249	t501	u120	u250	u500	u1000	Avrg.
Theo	20	40	83	167	49,05	101,55	201,2	400,55	132,79
RLM-LACC	21	42	89	176,67	49,67	105,33	209	420,33	139,13
1PTX FF2	20,95	41	84	168	49,05	101,7	201,3	400,65	133,33
LIMX FF1	21	41	84	168,85	49,1	101,8	202,35	417,5	135,7
UX FFD2	21	41	84,15	169,8	49,05	101,75	201,5	401,5	133,72
MUX FFD1	21	41	84,05	169,2	49,05	101,65	201,3	401,5	133,59
HGGA	20,1	40	83	167	49,15	101,7	201,2	400,55	132,84
MTP	21,55	44,1	90,45	181,85	49,15	102,15	203,4	404,55	137,15

According to the results RLM–LACC performs statistically better than SRAN–GDEL in all problem instances. SRAN–IEQ and RLM–IEQ have only one problem instances that has similar performance with RLM–LACC. RLM–GDEL has three and RL–IEQ has two and RL–GDEL has four problem instances that have similar performance. Late acceptance heuristics have better results statistically compared to other move acceptance methods. All results show that RLM–LACC has significantly better than other hyper-heuristics. Its average best results are compared with Falkanauer’s HGGA [7], Martello and Toth’s [76] reduction algorithm (MTP) and 4 different genetic operator used genetic algorithms [59], these are namely 1PTX FF2, LIMX FF1, UX FFD2 and MUX FFD1. The results are shown in Table 5.20

RLM–LACC has worse average best results compared to other studies. It is competitive only in *t60* and *u120* data sets. RLM–LACC has better performance than Martello and Toth’s reduction algorithm in the datasets which are created as triplets (*t60*, *t120*, *t249* and *t501*).

6. CONCLUSION

In the thesis, the performance of grouping hyper-heuristics on well-known multi-objective and single-objective grouping problems are investigated. Three multi-objective grouping problems, graph coloring, exam timetabling and data clustering and one single-objective grouping problem, bin packing are used in the study. As a total of 9 hyper-heuristics are used combining 3 heuristic selection methods; simple random, reinforcement learning, modified reinforcement learning and 3 move acceptance methods; great deluge, improve or equal, late acceptance. 10 low-level heuristics are implemented and 3 different hyper-heuristic framework types for multi-objective optimization are used.

In multi-objective hyper-heuristics, the tuning experiments show that tournament size does not have much effect on the performance of hyper-heuristics and tournament size 2 performs slightly better. The choice of the pareto front interval in the multi-objective hyper-heuristic solvers affects their average performance. A smaller interval generates a better average performance. The number of heuristics used in the multi-objective hyper-heuristics also affects the performance of the hyper-heuristics. The use of simpler and fast heuristics generates a better performance than the use of additional more time consuming intelligent heuristics within the grouping hyper-heuristic framework.

The hyper-heuristic framework types used for the multi-objective grouping problems differs how a candidate solution is selected from the list of non-dominating candidate solutions for applying a selected heuristic. However, the proposed methods, namely; cyclic and apply to all hyper-heuristic types do not give better results than the generic hyper-heuristic type which selects a candidate solution randomly from the pareto front. Also the modified reinforcement learning does not yield better results than the standard reinforcement learning for multi-objective problems. The number of accepted moves are too few, therefore the reinforcement learning mechanism loses its advantage of learning. The late acceptance move acceptance method has a slightly better performance than great deluge and improve or equal.

The tuning experiments show that the reinforcement learning with late acceptance and simple random with late acceptance deliver similar performances in finding the best solutions and they are slightly better than the other hyper-heuristics. The performances of these two hyper-heuristics are compared to the previously proposed approaches. The proposed hyper-heuristics have very successful results in graph coloring and exam timetabling. They succeeded to find the expected best solutions for the benchmark problem instances. They perform even better than some of them or deliver a matching performance. However, in data clustering, proposed hyper-heuristics failed to find any feasible solution in three of the four real-world data sets. On the other hand, they successfully find the expected best solutions one real-world and four synthetic problem instances.

As for the single objective grouping problem, similarly, hyper-heuristics that use late acceptance move acceptance method have better performances compared to others. Modified reinforcement learning has a slightly better performance than reinforcement learning for bin packing. Modified reinforcement learning with late acceptance is significantly better than the other 8 hyper-heuristics. This hyper-heuristic is compared to the previous approaches and it delivers better results than Martello and Toth's reduction algorithm for the triplet instances. It has competitive results for the small problem instances but when the problem size gets bigger, the results of proposed hyper-heuristic worsens. Considering that the framework only allows feasible solutions for bin packing, it is likely that hyper-heuristic cannot learn which heuristic delivers a good performance given the time limit for a run, since some of the improvement attempts and hence the relevant steps are ignored due to the repair mechanism.

The results of the experiments is consistent with the previous findings [33, 39, 78]. Learning during the heuristic selection process definitely helps, but move acceptance plays a major role in the performance of hyper-heuristics. This is possibly because the small number of heuristics used within the hyper-heuristics. A goal of hyper-heuristics is raising the level of generality by automating the heuristic design process. With this in mind, selection hyper-heuristics based on a general framework even including the low level of heuristics for solving grouping problems are investigated in this thesis. The grouping selection hyper-heuristic frameworks are designed assuming a domain barrier between the high level hyper-heuristic methodology and the problem domain dependent components such as low level heuristics.

Only problem independent information, such as fitness is allowed to pass across these layers. Using this information, selection hyper-heuristics manage the low level heuristics and choose the best alternative at each iterative step during the search process. In the overall, the empirical results show that the grouping selection hyper-heuristics provide although not the best, but competitive and promising results without compromising from the solution quality. In the future, more low-level heuristics can be tested along with new hyper-heuristic methodologies and the most efficient low-level heuristic subset can be identified to improve the performance of hyper-heuristics. The grouping hyper-heuristics can be tested with other multi-objective and single-objective grouping problems.

REFERENCES

1. Burke, E. K., M. Gendraue, M. Hyde, G. Kendall, G. Ochoa, E. Özcan and R. Qu, “Hyper-heuristics: A Survey of the State of the Art, to appear in JORS.”, .
2. Burke, E. K., M. Hyde, G. Kendall, G. Ochoa, E. Özcan and J. Woodward, *Handbook of Metaheuristics*, chap. A Classification of Hyper-heuristic Approaches, International Series in Operations Research & Management Science, Springer, 2009.
3. Özcan, E., B. Bilgin and E. E. Korkmaz, “A Comprehensive Survey of Hyperheuristics”, *Intelligent Data Analysis*, vol. 12, no. 1, pp. 1–21, 2008.
4. Falkenauer, E., *Genetic Algorithms and Grouping Problems*, John Wiley and Sons, 1998.
5. Carter, M. W. and G. Laporte, “Recent Developments in Practical Examination Timetabling”, in *Practice and Theory of Automated Timetabling*, pp. 3–21, 1995.
6. Johnson, D. J. and M. A. Trick (editors), *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*, American Mathematical Society, Boston, MA, USA, 1996, ISBN 0821866095.
7. Falkenauer, E., “A hybrid grouping genetic algorithm for bin packing”, *Journal of Heuristics*, vol. 2, pp. 5–30, 1996.
8. Burke, E. K., E. Hart, G. Kendall, J. Newall, P. Ross and S. Schulenburg, “Hyper-heuristics: An emerging direction in modern search technology”, in F. Glover and G. Kochenberger (editors), *Handbook of Metaheuristics*, pp. 457–474, Kluwer, 2003.
9. Burke, E. K., M. Hyde, G. Kendall, G. Ochoa, E. Özcan and J. Woodward, “Exploring Hyper-heuristic Methodologies with Genetic Programming”, in C. Mumford and L. Jain (editors), *Computational Intelligence: Collaboration, Fusion and Emergence*, Intelligent Systems Reference Library, pp. 111–111, Springer, 2009, ISBN 978-3-642-

01798-8.

10. Ross, P., “Hyper-heuristics”, in E. K. Burke and G. Kendall (editors), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chap. 17, pp. 529–556, Springer, 2005.
11. Denzinger, J., M. Fuchs and M. Fuchs, “High Performance ATP Systems by Combining Several AI Methods”, in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, pp. 102–107, 1997.
12. Cowling, P., G. Kendall and E. Soubeiga, “A hyperheuristic approach for scheduling a sales summit”, in *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, Lecture Notes in Computer Science, pp. 176–190, Springer, Konstanz, Germany, August 2000.
13. Burke, E. K., G. Kendall and E. Soubeiga, “A Tabu-Search Hyperheuristic for Timetabling and Rostering”, *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.
14. Chakhlevitch, K. and P. I. Cowling, “Hyperheuristics: Recent Developments”, in C. Cotta, M. Sevaux and K. Sörensen (editors), *Adaptive and Multilevel Metaheuristics*, vol. 136 of *Studies in Computational Intelligence*, pp. 3–29, Springer, 2008, ISBN 978-3-540-79437-0.
15. Kendall, G. and M. Mohamad, “Channel Assignment In Cellular Communication Using A Great Deluge Hyper-Heuristic”, in *Proceedings of the 2004 IEEE International Conference on Network (ICON2004)*, pp. 769–773, Singapore, 16-19 November 2004.
16. Kendall, G. and M. Mohamad, “Channel Assignment Optimisation Using a Hyper-heuristic”, in *Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent Systems (CIS2004)*, pp. 790–795, Singapore, 1-3 December 2004.
17. Ayob, M. and G. Kendall, “A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine”, in *Proceedings of the*

International Conference on Intelligent Technologies (InTech'03), pp. 132–141, Chiang Mai, Thailand, December 17-19 2003.

18. Cowling, P. and K. Chakhlevitch, “Hyperheuristics for Managing a Large Collection of Low Level Heuristics to Schedule Personnel”, in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC2003)*, pp. 1214–1221, IEEE Computer Society Press, Canberra, Australia, 2003.
19. Han, L. and G. Kendall, “Guided Operators for a Hyper-Heuristic Genetic Algorithm”, in *Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence (AI 03)*, pp. 807–820, Perth, Australia, 3-5 December 2003.
20. Bai, R., J. Blazewicz, E. K. Burke, G. Kendall and B. McCollum, “A Simulated Annealing Hyper-Heuristic Methodology for Flexible Decision Support”, Tech. Rep. NOTTCS-TR-2007-8, School of CSiT, University of Nottingham, 2007.
21. Dowsland, K. A., E. Soubeiga and E. K. Burke, “A Simulated Annealing Hyper-Heuristic for Determining Shipper Sizes”, *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.
22. Nareyek, A., “Choosing Search Heuristics by Non-Stationary Reinforcement Learning”, in M. G. C. Resende and J. P. de Sousa (editors), *Metaheuristics: Computer Decision-Making*, chap. 9, pp. 523–544, Kluwer, 2003.
23. Antunes, C. H., P. Lima, E. Oliveira and D. F. Costa, “A Multi-Objective Simulated Annealing Approach to Reactive Power Compensation”, in *Proceedings of the 8th Metaheuristic International Conference*, 2009.
24. Burke, E. K., J. D. Landa-Silva and E. Soubeiga, *Meta-heuristics: Progress as Real Problem Solvers*, chap. Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, pp. 129–158, Springer, 2005.

25. Bai, R. and G. Kendall, “An investigation of automated planograms using a simulated annealing based hyper-heuristics”, in T. Ibaraki, K. Nonobe and M. Yagiura (editors), *Metaheuristics: Progress as Real Problem Solver - (Operations Research/Computer Science Interface Series, Vol.32)*, pp. 87–108, Springer, 2005.
26. Bai, R., E. K. Burke and G. Kendall, “Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation”, *Journal of the Operational Research Society*, vol. 59, pp. 1387 – 1397, 2008.
27. Burke, E. K., G. Kendall, J. D. Landa-Silva, R. O’Brien and E. Soubeiga, “An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem”, in *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2263–2270, Edinburgh, Scotland, September 2-5 2005.
28. Bilgin, B., E. Özcan and E. E. Korkmaz, “An Experimental Study on Hyper-Heuristics and Final Exam Scheduling.”, in *Proc. of the International Conference on the Practice and Theory of Automated Timetabling (PATAT’06)*, pp. 123–140, 2006.
29. Chen, P. C., G. Kendall and G. Vanden-Berghe, “An Ant Based Hyper-heuristic for the Travelling Tournament Problem”, in *Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched 2007)*, pp. 19–26, Hawaii, 2007.
30. Pisinger, D. and S. Ropke, “A general heuristic for vehicle routing problems”, *Computers and Operations Research*, vol. 34, pp. 2403– 2435, 2007.
31. Cowling, P., G. Kendall and E. Soubeiga, “Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation”, in S. Cagani, J. Gottlieb, E. Hart, M. Middendorf and R. Goenther (editors), *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, vol. 2279 of *Lecture Notes in Computer Science*, pp. 1–10, Springer-Verlag, Kinsale, Ireland, April 3-4 2002.
32. Chakhlevitch, K. and P. Cowling, “Choosing the Fittest Subset of Low Level Heuristics in a Hyperheuristic Framework”, in *Proceedings of 5th European Conference on*

Evolutionary Computation in Combinatorial Optimization (EvoCOP2005), vol. 3448 of *Lecture Notes in Computer Science*, pp. 25–33, Springer, 2005.

33. Özcan, E., Y. Bykov, M. Birben and E. K. Burke, “Examination Timetabling Using Late Acceptance Hyper-heuristics”, in *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 997–1004, 2009.
34. Kaelbling, L. P., M. L. Littman and A. W. Moore, “Reinforcement learning: a survey”, *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
35. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.
36. Dueck, G., “New Optimization Heuristics: The Great Deluge Algorithm and the Record-to Record Travel”, *Journal of Computational Physics*, vol. 104, pp. 86–92, 1993.
37. Özcan, E., B. Bilgin and E. E. Korkmaz, “Hill Climbers and Mutational Heuristics in Hyperheuristics”, in *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, vol. 4193 of *Lecture Notes in Computer Science*, pp. 202–211, Reykjavik, Iceland, September 2006.
38. Bai, R. and G. Kendall, “A Model for Fresh Produce Shelf-Space Allocation and Inventory Management with Freshness-Condition-Dependent Demand”, *INFORMS Journal on Computing*, vol. 20, no. 1, pp. 78–85, 2008.
39. Burke, E. K., G. Kendall, M. M. Sır and E. Özcan, “Monte Carlo hyper-heuristics for examination timetabling”, in *Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, 2008.
40. Swan, J., E. Özcan and G. Kendall, “Hyperion - A Recursive Hyper-heuristic Framework”, in C. A. C. Coello (editor), *Learning and Intelligent Optimization, 5th International Conference, LION 5*, LNCS, 2011.
41. Burke, E., T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic and

- J. Vazquez-Rodriguez, “Hyflex: A flexible framework for the design and analysis of hyper-heuristics.”, in *Proceedings of the Multidisciplinary International Scheduling Conference (MISTA09)*, pp. 790–797, 2009.
42. Burke, E. K., T. Curtois, M. R. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J. A. V. Rodríguez and M. Gendreau, “Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms”, in *IEEE Congress on Evolutionary Computation*, pp. 1–8, 2010.
43. Osyczka, “Multicriteria optimization for engineering design”, , 1985.
44. Miettinen, K., M. M.Mkel, P. Neittaanmaki and J. Priaux, *Evolutionary Algorithms in Engineering and Computer Science*, chap. Evolutionary Multi-criterion Optimization, WILEY, 1999.
45. Schaffer, J. D., “Multiple objective optimization with vector-evaluated genetic algorithms”, in *Int’l Conf. on Genetic Algorithms and their Applications, Carnegie-Mellon Univ., Pittsburgh*, 1985.
46. Fonseca, C. M. and P. J. Fleming, “Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization”, , 1993.
47. Horn, J., N. Nafpliotis and D. E. Goldberg, “A niched Pareto genetic algorithm for multi-objective optimization”, in *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 82–87, 1994.
48. Zitzler, E. and L. Thiele, “An Evolutionary Approach for Multiobjective Optimization: The Strength Pareto Approach”, TIK Report 43, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, May 1998.
49. Srinivas, N. and K. Deb, “Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms”, *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.
50. Marler, R. T. and J. S. Arora, “Survey of multi-objective optimization methods for

- engineering”, *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369–395, 2004.
51. Chinchuluun, A. and P. M. Pardalos, “A survey of recent developments in multiobjective optimization”, *Annals of Operations Research*, vol. 154, pp. 29–50, 2007.
 52. Andersson, J., “A survey of multiobjective optimization in engineering design”, , 2000.
 53. Du, J., E. E. Korkmaz, R. Alhajj and K. Barker, “Novel Clustering That Employs Genetic Algorithm with New Representation Scheme and Multiple Objectives.”, in Y. Kambayashi, M. K. Mohania and W. W (editors), *DaWaK*, vol. 3181 of *Lecture Notes in Computer Science*, pp. 219–228, Springer, 2004.
 54. Radcliffe, N. J., “Formal Analysis and random respectful recombination”, in *Proceedings of the 4th International Conference on Genetic Algorithm*, pp. 222–229, 1991.
 55. Korkmaz, E. E., “A Two-Level Clustering Method Using Linear Linkage Encoding”, , 2006.
 56. Ülker, O., E. Özcan and E. E. Korkmaz, “Linear Linkage Encoding in Grouping Problems: Applications on Graph Coloring and Timetabling”, in *International Conference on the Practice and Theory of Automated Timetabling*, 2006.
 57. Burke, E. K. and Y. Bykov, “A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems”, in *PATAT 2008 Conference, Montreal, Canada*, 2008.
 58. Yılmaz, B. and E. E. Korkmaz, “Representation issue in graph coloring”, in *Intelligent Systems Design and Applications*, pp. 1171–1176, 2010.
 59. Ülker, O., E. E. Korkmaz and E. Özcan, “A Grouping Genetic Algorithm Using Linear Linkage Encoding for Bin Packing”, in *Parallel Problem Solving from Nature*, pp. 1140–1149, 2008.

60. Jain, A. K., M. N. Murty and P. J. Flynn, “Data clustering: a review”, *ACM Computing Surveys*, vol. 31, pp. 264–323, 1999.
61. Cole, A. J., “The preparation of examination timetables using a small-store computer”, *The Computer J.*, vol. 7, pp. 117–121, 1964.
62. Broder, S., “Final examination scheduling”, *Communications of The ACM*, vol. 7, pp. 494–498, 1964.
63. Wood, D. C., “A system for computing university examination timetables”, *The Computer Journal*, vol. 11, pp. 41–47, 1968.
64. Carter, M. W., “OR Practice—A Survey of Practical Applications of Examination Timetabling Algorithms”, *Operations Research*, vol. 34, pp. 193–202, 1986.
65. Leighton, F. T., “A Graph Coloring Algorithm for Large Scheduling Problems”, in *Journal of Research of the National Bureau of Standards*, pp. 489–506, 1979.
66. Burke, E. K. and J. P. Newall, “A multi-stage evolutionary algorithm for the timetable problem”, *IEEE Transactions on Evolutionary Computation*, 1998.
67. Paquete, F. and C. M. Fonseca, “A Study of Examination Timetabling with Multiobjective Evolutionary Algorithms”, , 2001.
68. Özcan, E., “Towards an XML based standard for Timetabling Problems: TTML”, , 2005.
69. Cheong, C. Y., K. C. Tan and B. Veeravalli, “Solving the Exam Timetabling Problem via a Multi-Objective Evolutionary Algorithm - A More General Approach”, , 2007.
70. Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.
71. Avanthay, C., A. Hertz and N. Zufferey, “A variable neighborhood search for graph coloring”, *European Journal of Operational Research*, vol. 151, pp. 379–388, 2003.

72. Davis, T., “The handbook of genetic algorithms”, , 1991.
73. Hertz, A. and D. D. Werra, “Using Tabu Search Techniques for Graph Coloring”, *Computing*, 1987.
74. Johnson, D. S., C. R. Aragon, L. A. McGeoch and C. Schevon, “Optimization by simulated annealing: an experimental evaluation; part ii”, *Operational Research*, 1991.
75. Co Man, E. G., M. R. Garey and D. S. Johnson, “Bin packing approximation algorithms: A survey”, , 1996.
76. Martello, S. and P. Toth, “Lower bounds and reduction procedures for the bin packing problem”, *Discrete Applied Mathematics*, vol. 28, no. 1, pp. 59–70, 1990.
77. Falkenauer, E. and A. Delchambre, “A Genetic Algorithm for Bin Packing and Line Balancing”, in *International Conference on Robotics and Automation*, 1992.
78. Özcan, E., M. Mısıır, G. Ochoa and E. K. Burke, “A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling”, *Int. J. of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, 2010.
79. Qu, R., E. K. Burke, B. Mccollum, L. T. G. Merlot and S. Y. Lee, “A survey of search methodologies and automated system development for examination timetabling”, *Journal of Scheduling*, vol. 12, pp. 55–89, 2009.
80. Külahçiođlu, B., *Multiobjective Hyperheuristic for Data Clustering and Linear Linkage Encoding*, Master’s thesis, Yeditepe University, 2007.
81. Brucker, P., *Scheduling Algorithms*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edn., 2001, ISBN 3540415106.
82. Kirovski, D. and M. Potkonjak, “Efficient Coloring of a Large Spectrum of Graphs”, in *35th Design Automation Conference Proceedings*, pp. 427–432, 1998.

83. Carter, M. W., G. Laporte and S. T. Lee, “Examination timetabling: algorithmic strategies and applications.”, *Journal of the Operational Research Society*, vol. 47, pp. 373–383, 1996.
84. Caramia, M., P. Dell’Olmo and G. F. Italiano, “New algorithms for examination timetabling”, in *Algorithm Engineering 4th International Workshop 2000*, Lecture Notes in Computer Science 1982, pp. 230–241, Springer-Verlag, Berlin Heidelberg New York, September 2001.
85. Merlot, L. T. G., N. Boland, B. D. Hughes and P. J. Stuckey, “A Hybrid Algorithm for the Examination Timetabling Problem.”, in P. Burke, E.; De Causmaecker (editor), *Proceedings of Practice and Theory of Automated Timetabling, Fourth International Conference*, pp. 348–371, Gent, Belgium, August 2002.