UNITED KINGDOM · CHINA · MALAYSIA

University of Nottingham

Multi-stage Hyper-heuristics for Optimisation Problems

Ahmed Kheiri, B.Sc. (Hons), M.Sc.

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

December 2014

Abstract

There is a growing interest towards self configuring/tuning automated general-purpose reusable heuristic approaches for combinatorial optimisation, such as, hyper-heuristics. Hyper-heuristics are search methodologies which explore the space of heuristics rather than the solutions to solve a broad range of hard computational problems without requiring any expert intervention. There are two common types of hyper-heuristics in the literature: *selection* and *generation* methodologies. This work focusses on the former type of hyper-heuristics. Almost all selection hyper-heuristics perform a single point based iterative search over the space of heuristics by selecting and applying a suitable heuristic to the solution in hand at each decision point. Then the newly generated solution is either accepted or rejected using an acceptance method. This improvement process is repeated starting from an initial solution until a set of termination criteria is satisfied. The number of studies on the design of hyper-heuristic methodologies has been rapidly increasing and currently, we already have a variety of approaches, each with their own strengths and weaknesses. It has been observed that different hyper-heuristics perform differently on a given subset of problem instances and more importantly, a hyper-heuristic performs differently as the set of low level heuristics vary. This thesis introduces a general "multi-stage" hyper-heuristic framework enabling the use and exploitation of multiple selection hyper-heuristics at different stages during the search process. The goal is designing an approach utilising multiple hyper-heuristics for a more effective and efficient overall performance when compared to the performance of each constituent selection hyper-heuristic. The level of generality that a hyper-heuristic can achieve has always been of interest to the hyper-heuristic researchers. Hence, a variety of multi-stage hyper-heuristics based on the framework are not only applied to the real-world combinatorial optimisation problems of high school timetabling, multi-mode resource-constrained multi-project scheduling and construction of magic squares, but also tested on the well known hyper-heuristic benchmark of CHeSC 2011. The empirical results show that the multi-stage hyper-heuristics designed based on the proposed framework are still inherently general, easy-to-implement, adaptive and reusable. They can be extremely effective solvers considering their success in the competitions of ITC 2011 and MISTA 2013. Moreover, a particular multi-stage hyper-heuristic outperformed the state-of-the-art selection hyper-heuristic from CHeSC 2011.

Acknowledgements

I would like to express my profound gratitude to Almighty God for all the physical and mental strength He bestowed on me during all the activities of the project and finally during the preparation of this thesis.

I am deeply grateful to Dr. Ender Özcan, my supervisor who generously rendered help and encouragement during the process of this thesis. Whatever I have accomplished in pursuing this undertaking is due to his guidance and thoughtful advice. To him my thanks.

I have no doubt that this thesis is stronger because of the collaborative working I have had with Dr. Andrew J. Parkes, Dr. Daniel Karapetyan, Shahriar Asta, Leena N. Ahmed, Mohamed Elsayed, Dr. Moh'd Khaled Yousef Shambour and Dr. Mustafa Mısır.

I would like to thank the examiners, Professor Greet Vanden Berghe and Professor Robert John, for their valuable comments.

Also, I am grateful to all members of the Automated Scheduling, Optimisation and Planning (ASAP) research group for all what I have learnt from them.

My greatest debt is, as always, to my parents, brothers, sisters and my lovely wife who persistently rendered me the much needed support in my academic endeavours.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Currently, most of the decision support systems are developed by experts and they are often custom tailored to specific application domains. Hence, they cannot be reused for solving a problem from another domain. Such systems are expensive to build and maintain. Even a slight change in the same problem could require expert intervention. On the other hand, there has been some significant scientific progress in developing automated general purpose systems that can learn, adapt and improve their behaviour on the fly while solving a given problem. Such systems are applicable to different instances from not only the same domain but also other problem domains. Moreover, they are easy to build and maintain and so less costly. Hyper-heuristics have emerged as such general purpose high level search methodologies. A *hyper-heuristic* performs a search over the space formed by a fixed set of (meta-)heuristics which operate directly on the space of solutions, for solving computationally hard problems [2–8]. A class of hyper-heuristics aims to provide solutions across a range of problem domains by processing problem independent information obtained from selecting/mixing a set of low level heuristics which operate at the problem level during the search process. There are two main types of hyper-heuristics in the academic literature: methodologies used for *generation* or *selection* of heuristics [2, 4, 9]. Even though the term hyper-heuristic was coined and introduced recently, the idea of combining the different existing heuristics (neighbourhood operators) with the goal of exploiting their strengths dates back to the early 1960s [10, 11]. There has been a growing interest in hyper-heuristics since then. This study focuses on an iterative selection hyper-heuristic framework based on a single point search [8, 12].

Almost all previously proposed selection hyper-heuristics are designed respecting the concept of a *domain barrier* which separates the hyper-heuristic from the problem domain containing the low level heuristics [12] as illustrated in Figure 1.1. In this framework, the hyper-heuristic layer interacts with the problem domain and heuristic layers through problem independent measures, such as the quality change in a candidate solution when the selected heuristic is employed. Traditionally, the barrier prohibits any problem domain specific information to pass through to the hyper-heuristic level. This type of layered and modular approach to the design of automated search methodologies supports the development of more general methods than currently there exist, which are applicable to unseen instances from a single problem domain or even different problem domains. Moreover, reuse of algorithmic components becomes possible.

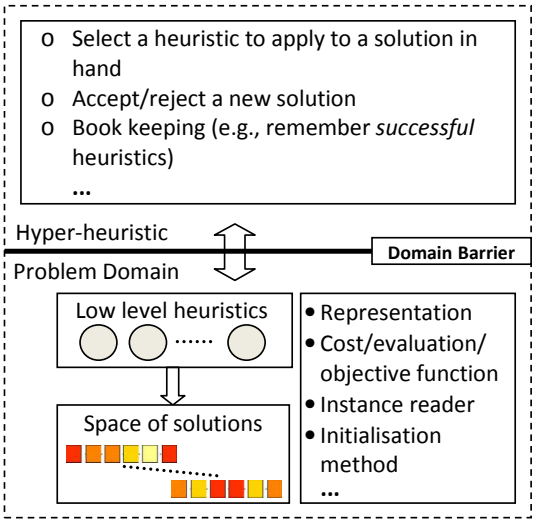


FIGURE 1.1: A selection hyper-heuristic framework and the domain barrier

A selection hyper-heuristic is a high level problem solving framework that can select and apply an appropriate low level heuristic used to modify a solution in hand at each decision point, given a particular problem instance and a number of low level heuristics. The selection hyper-heuristic framework combines two essential components: *heuristic selection* and *move acceptance* methods processes under a single point search framework, as identified in [3, 7, 8, 12, 13]. Such a hyper-heuristic attempts to improve a solution (or solutions) by selecting a perturbative or constructive heuristic (or heuristics), each processing and returning a complete or partial solution when invoked, followed by a decision to accept or reject the resulting solution at a given step during the search process. The solution is evaluated according to the objective value (also referred as cost

or fitness in this thesis). The search process continues iteratively until the termination criteria are satisfied. Finally, the best solution at hand is returned for a given problem.

Many researchers and practitioners have been progressively involved in hyper-heuristic studies for solving difficult real world combinatorial optimisation problems ranging from channel assignment to production scheduling (Table 1.1).

TABLE 1.1: Some selected problem domains in which hyper-heuristics were used as solution methodologies

Problem Domain [Reference]	Problem Domain [Reference]
Channel assignment [14]	Job shop scheduling [10]
Component placement sequencing [15]	Sales summit scheduling [12]
Examination timetabling [16]	Space allocation [17]
Nurse rostering [18]	University course timetabling [18]
Orc quest, logistics domain [19]	Vehicle routing problems [20]
Packing [21]	Production scheduling [22]

1.1 Research Motivation and Contributions

Although Cowling et al. [12] implied that hyper-heuristics are problem independent approaches, there is strong empirical evidence showing that the choice of selection hyper-heuristic components influences its overall performance [13]. Bilgin et al. [23] showed that different combinations of selection hyper-heuristic components yield different performances on examination timetabling problems. Özcan et al. [8] showed that the move acceptance is more influential on the performance of a selection hyper-heuristic if the number of low level heuristics is low and they are mutational. Then, the choice of move acceptance component becomes more crucial. A recent theoretical study showed that mixing heuristics could lead to exponentially faster search than using each standalone heuristic on some benchmark functions [24]. These observations are crucial, since they imply that another level can be introduced on top of the hyper-heuristics for managing them. Then the question arises: "How are we going to end this hierarchical growth in the levels?".

This situation generates a curiosity and points out a potential modification of the standard single point-based search framework of selection hyper-heuristics. Multi-stage operation of multiple selection hyper-heuristics needs to be supported. Then, another hyper-heuristic level needs to be introduced for managing multiple hyper-heuristics which operate on top of low level heuristics. Considering the recursive hyper-heuristic definition

of 'heuristics to choose heuristics' [12], those hyper-heuristics could require another level and so on, causing a hierarchical growth in the hyper-heuristic levels like a tree. An attempt was made in [25] to flatten the hierarchical growth in the hyper-heuristic levels for the move acceptance via group decision making strategies. The authors suggest combining multiple move acceptance strategies under a group decision making strategy. The performance of four such group decision making move acceptance methods are analysed within different hyper-heuristics over a set of benchmark functions. The experimental results show that the group decision making strategies have potential to significantly improve the overall performance of selection hyper-heuristics.

In another study, Özcan and Burke [26] provided a general *multi-level hierarchical search* methodology by taking advantage of the recursive nature of the hyper-heuristics definition, 'heuristics which search the space of heuristics'. Their proposal suggests an additional level on top of a hyper-heuristic in order to make use of different types of heuristics (operators) in cooperation. One of the frameworks proposed in [13] that handle a set of mutational and hill climber heuristics by invoking a mutational heuristic first followed by a hill climber, is actually performing a multi-level search. The higher level in this framework is managing two different hyper-heuristics allowing to employ diversifying (exploring of the search space) and intensifying (exploiting the accumulated search experience) phases. The effectiveness of the multi-level search framework that combines multiple selection hyper-heuristics suggested by Özcan and Burke [26] has not been investigated further.

This thesis describes a multi-level framework which allows the use of multiple hyper-heuristics during the search process. Given that one of the selection hyper-heuristics would be employed at each stage during the search process, we will refer to the overall approach as *multi-stage (selection) hyper-heuristic*. The additional level on top of multiple selection hyper-heuristics will be referred to as *multi-stage level*. The proposed multi-stage hyper-heuristic framework is general, reusable and useful in relieving the difficulty of choosing a hyper-heuristic method for solving a problem, by automating the process of selecting a hyper-heuristic at different point of the search process. Five novel multi-stage selection hyper-heuristics based on the framework is designed, implemented and analysed in this thesis:

- MSHH1: A greedy heuristic selection strategy referred to as *dominance-based* heuristic selection method reducing the number of low level heuristics considering the trade-off between improvement achieved by a low level heuristic with a

given setting and the step it takes to achieve that performance, is combined with the random descent selection hyper-heuristic.

- MSHH2: The *Robinhood* (**Ro**und-ro**bin** neighbour**hood**) hyper-heuristic contains a heuristic selection component that allocates equal share from the overall execution time for each low level heuristic, while the different combined move acceptance criteria enable partial restarts when the search process stagnates.

- MSHH3: The *HySST* (**Hy**per-heuristic **S**earch **S**trategies and **T**imetabling) hyper-heuristic switches between diversification and intensification processes automatically and allows partial restarts via a threshold move acceptance method whose parameter is also controlled by the proposed method.

- MSHH4: A multi-stage selection hyper-heuristic approach that extends the three approaches briefly described above. It uses the dominance-based method in MSHH1 combined with the round-robin strategy in MSHH2 to reduce the set of low level heuristics, and also applies a modified version of the move acceptance method used in MSHH3.

- MSHH5: A multi-stage selection hyper-heuristic approach that extends the previous multi-stage hyper-heuristic (MSHH4) and makes use of the *relay hybridisation* [27] technique which applies a low level heuristic to a solution generated by applying a preceding heuristic.

Additionally, the technique used in [28, 29] is implemented as MSHH6 and applied to a benchmark to test its level of generality.

It is always of interest to researchers and practitioners to find the state-of-the-art approach for solving a particular problem. Although hyper-heuristic research aims for the level of generality, still knowing the relative position of hyper-heuristics with respect to the state-of-the-art for a specific problem domain would be useful. One way of establishing state-of-the-art for a specific problem is through competitions/challenges. Therefore, Some of the multi-stage hyper-heuristics, briefly described above, are initially designed for solving some specific problems. This has indeed required design and implementation of the problem domain layer components, such as domain specific low level heuristics.

High school timetabling is a well-known real-world combinatorial optimisation problem. The problem is known to be NP-complete [30] even in simplified forms. It requires the scheduling (assignment) of events and resources, such as courses, classes of students,

teachers, rooms and more within a fixed number of time slots subject to a set of constraints. A competition has been organised on high school timetabling: the Third International Timetabling Competition (ITC 2011) to form a benchmark for future studies, determine the state-of-the-art solution method and promote researchers and practitioners to deal with the problems as they are without discarding the real world complexities. MSHH3 is designed for high school timetabling problem and has entered into the three Rounds of the competition. This multi-stage hyper-heuristic has generated the best new solutions for three given instances in Round 1 and gained the second place in Rounds 2 and 3 at ITC 2011.

Project scheduling is a common real world optimisation problem, and has been addressed by a competition organised together with the MISTA 2013 conference. In the competition, multiple projects had to be scheduled whilst taking into account the availability of local and global resources under a set of constraints. MSHH4 is designed for solving the multi-mode resource-constrained multi-project scheduling problem and eventually has won the competition becoming the state-of-the-art in this domain.

A magic square is a square matrix that contains distinct numbers in which the summation of the numbers in each row, column and the two diagonals has the same constant total known as the magic number [31]. Constructing the magic square has been considered as a hard computational problem domain [32]. This was topic of a competition and MSHH5 is used as a method to solve the constrained-version of magic squares.

In order to show that the proposed multi-stage selection hyper-heuristics are sufficiently general and can be applied to different problem domains without requiring any change, they are implemented as extension to a public software library, referred to as HyFlex (Hyper-heuristics Flexible framework) which provides an interface for the implementation of not only hyper-heuristics but also other (meta)heuristics and problem domains. HyFlex has been used for benchmarking of hyper-heuristics [33]. The experimental results indicate the success of the multi-stage selection hyper-heuristics designed under the proposed multi-stage framework. The framework is more general in the sense that it is applicable to a variety of problems and more effective than the existing approaches which frequently ignore the real-world complexities. A selection hyper-heuristic should be "(1) fast to implement, (2) requiring far less expertise in either the problem domain or heuristic methods, and (3) robust enough to effectively handle a range of problems" [12]. The proposed multi-stage hyper-heuristic framework in this thesis satisfies all previously suggested design criteria.

1.2 Structure of Thesis

The thesis is structured as follows:

- Chapter 1: Introduces the thesis topic and relevant concepts.
- Chapter 2: Provides the background and literature survey. Selection hyper-heuristics, problem domains dealt with in this work and the previous approaches used to solve those problem domains are summarised.
- Chapter 3: Introduces the problem domains in more detail, including the characteristics of the instances, low level heuristics and initialisation methods for each domain.
- Chapter 4: Describes the novelty and all algorithmic components of multi-stage hyper-heuristic framework. Additionally, several multi-stage hyper-heuristics to control a set of perturbative low level heuristics are explained.
- Chapter 5: Presents the competition results of the ITC 2011 and MISTA 2013 competitions, compares the performance of the different proposed multi-stage hyper-heuristics on HyFlex problems as well as constructing constrained-version of the magic-square problem; and reports the generality level of selection hyper-heuristics. The chapter provides the computational results and discussions.
- Chapter 6: Presents the conclusions of the research outcome and points out some future research directions.

1.3 Academic Publications Produced

The following academic articles, conference papers and extended abstracts have been produced as a result of this research. It is worthy to mention that the results and the performance analysis of the multi-stage hyper-heuristics which have been published on academic papers are revised in this thesis and additional results obtained from the testing on the different problem domains are reported.

- Ahmed Kheiri and Ender Özcan. Constructing constrained-version of magic squares using selection hyper-heuristics. The Computer Journal, 57(3):469-479, 2014. [journal]

- Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing selection hyper-heuristic. Soft Computing, 17(12):2279-2292, 2013. [journal]

- Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. A stochastic local search algorithm with adaptive acceptance for high-school timetabling. Annals of Operations Research, published online. [journal]

- Ahmed Kheiri and Ender Özcan. A multi-stage selection hyper-heuristic, under review. [journal]

- Ender Özcan, Mustafa Misir, and Ahmed Kheiri. Group decision making in selection hyper-heuristics, under review. [journal]

- Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem, under review. [journal]

- Leena N. Ahmed, Ender Özcan, and Ahmed Kheiri. Solving high school timetabling problems worldwide using selection hyper-heuristics, under review. [journal]

- Ahmed Kheiri and Ender Özcan. A hyper-heuristic with a round robin neighbourhood selection. In Martin Middendorf and Christian Blum, editors, Evolutionary Computation in Combinatorial Optimization, volume 7832 of Lecture Notes in Computer Science, pages 1-12. Springer Berlin Heidelberg, 2013. [conference]

- Mohd Khaled Yousef Shambour, Ahamad Tajudin Khader, Ahmed Kheiri, and Ender Özcan. A two stage approach for high school timetabling. In Minho Lee, Akira Hirose, Zeng-Guang Hou, and Rhee Man Kil, editors, Neural Information Processing, volume 8226 of Lecture Notes in Computer Science, pages 66-73. Springer Berlin Heidelberg, 2013. [conference]

- Ender Özcan, Mustafa Misir, and Ahmed Kheiri. Group decision making hyper-heuristics for function optimisation. In 13th UK Workshop on Computational Intelligence (UKCI2013), pages 327-333. IEEE, 2013. [conference]

- Ender Özcan and Ahmed Kheiri. A hyper-heuristic based on random gradient, greedy and dominance. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, Computer and Information Sciences II, pages 557-563. Springer London, 2012. [conference]

- Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In 12th UK Workshop on Computational Intelligence (UKCI2012), pages 1-8. IEEE, 2012. [conference]

- Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. In Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA2013), Ghent, Belgium, pages 836-839, 2013. [extended abstract]

- Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. HySST: hyper-heuristic search strategies and timetabling. In Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT2012), pages 497-499, 2012. [extended abstract]

Additionally, there are two other papers currently being prepared for submission to journals. There is also a number of abstract submissions produced out of this study and presented at international conferences, including OR53, OR54, OR56, 3rd Student Conference on Operational Research (SCOR2012), 25th Conference of the European Chapter on Combinatorial Optimization (ECCO2012), and 20th Conference of the International Federation of Operational Research Societies (IFORS2014).

Chapter 2

Literature Review

This chapter briefly describes some meta-heuristics commonly used for combinatorial optimisation and presents a survey of selection hyper-heuristics. In addition, an overview of the approaches proposed previously for solving the different problems dealt with in this work are provided.

2.1 Meta-heuristics

Pearl [34] defined heuristic as an intelligent search strategy for computer problem solving. In the field of optimisation problems, a heuristic can be considered as an educated guess or a 'rule of thumb', which guides the computational search required for finding a solution. Although the heuristic algorithms are designed to speed up the process of discovering a solution, yet the optimal solution is not guaranteed to be obtained. Heuristics are often problem-dependent methods that work well for an instance of a problem may or may not be used to solve another instance of another problem or even the same problem. In contrast, a *meta-heuristic* is a high level methodology which performs a search for solving any computationally hard problem.

The term meta-heuristic was first used by Glover [35] to describe Tabu Search and has recently been defined by Sörensen and Glover [36] as: "A meta-heuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimisation algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimisation algorithm according to the guidelines expressed in such a framework."

Meta-heuristics can be broadly classified into population-based meta-heuristics, also called multi-point meta-heuristics, and single-solution meta-heuristics, also called single-point meta-heuristics. The population-based meta-heuristics, such as, Evolutionary Algorithms [37], consist of a collection of individual solutions which are maintained in a population while the single-solution meta-heuristics, such as, Tabu Search [35], differ from population-based in that they improve and maintain a single solution.

Meta-heuristics provide "guidelines" for designing heuristic optimisation algorithms [36]. The *exploration* (diversification), being able to jump to the other regions of the search space and *exploitation* (intensification), being able to perform local search within a limited region using accumulated experience, capabilities and maintaining the balance between them are important for a metaheuristic, influencing its performance. Different meta-heuristics have different ways of maintaining that balance. Metaheuristics have to be tailored for a specific problem domain and often, they are successful in obtaining high quality solutions for that domain. However, meta-heuristics being a subset of heuristics come with no guarantee for the optimality of the obtained solutions. Moreover, they cannot be used for solving an instance from another problem domain. The maintenance of meta-heuristics could be costly requiring expert intervention. Even a slight modification in the description of the problem could require maintenance. Almost all meta-heuristics have parameters and their performance could be sensitive to the setting of those parameters. There are automated parameter tuning methods, such as F-race [38], REVAC [39] and ParamILS [40] to overcome this issue. The parameter tuning process increases the overall computation time of an approach while searching for a high quality solution to a given problem instance. However, there could be a trade-off and a higher quality solutions could be obtained for a given problem in the expense of spending more time on tuning. A selected set of well known meta-heuristics, including Iterated Local Search, Tabu Search, Simulated Annealing, Great Deluge, Late Acceptance and Evolutionary Algorithms are briefly described in the following sections.

2.1.1 Iterated Local Search

Local search methods start from a candidate solution, and iteratively move from one solution to another from its direct neighbourhood [41]. The neighbourhood of a candidate solution is the set of solutions that can be generated by making (usually a small) change to the candidate solution. Hill climbing is a type of local search that iteratively improves the candidate solution by looking for a better solution to replace it, among the

ones in its neighbourhood. Although such methods are very easy and straightforward to be implemented, yet they can easily be trapped in a valley of local optima (local minimum), where the qualities of all neighbouring solutions are equal or worse. An iterated local search (ILS) method [42] is introduced to escape the search from a local minimum. In ILS, the current local minimum solution, generated by applying a local search, is perturbated (changed) leading to a new solution, then the local search is applied to the modified solution. This cycle of applying perturbation and local search methods is repeated until a termination criterion is satisfied. Iterated local search maintains the balance between the exploration and exploitation processes explicitly using perturbation and local search operators, respectively. [43]. An iterated local search algorithm is illustrated in Algorithm 1.

Algorithm 1: Pseudocode of the iterated local search method

```
1  Let S represent the candidate solution;
2  Let S_best represent the best solution;
3  S_initial ← CreateInitialSolution(); // generate random solution
4  S_best ← S_initial;
5  S ← LocalSearch(S_initial); // generate local optimal solution
6  repeat
7  |   S' ← Perturbation(S);
8  |   S'' ← LocalSearch(S');
9  |   S_best ← maintainBestSolution(S, S', S'');
10 |   if Accept(S, S'') then
11 |   |   S ← S'';
12 |   end
13 until TerminationCriterionSatisfied();
14 return S_best;
```

2.1.2 Tabu Search

Tabu search is a meta-heuristic introduced by Glover [35] back in the 1986. The basic idea of tabu search is to prevent the cyclic repetition of recent moves by maintaining a memory, called tabu list, to prevent visiting the recently visited solutions. This may help to guide the search away from local optima. The pseudocode of the tabu search method is given in Algorithm 2. Hyper-heuristics can employ similar strategy by preventing the selection of relatively poor performing heuristics.

Algorithm 2: Pseudocode of tabu search

```
1  Let S represent the candidate solution;
2  Let S_best represent the best solution;
3  S ← CreateInitialSolution(); // generate random solution
4  S_best ← S;
5  repeat
6  |  Generate(S');   /* generate neighbour solution S' that does not contain
   |  tabu elements */
7  |  UpdateTabuList(S');
8  |  S_best ← maintainBestSolution(S');
9  |  if S' isBetterThan S then
10 |  |  S ← S';
11 |  end
12 until TerminationCriterionSatisfied();
13 return S_best;
```

2.1.3 Simulated Annealing

Simulated Annealing (SA) [44] is a probabilistic meta-heuristic method, motivated by an analogy to the process of annealing in solids. At each step a new solution is generated. The new solution is accepted if it improved the previous solution. The non-improving solutions are accepted, to attempt escaping from local optimum, with a probability of $p = e^{-\frac{\Delta}{T}}$, where Δ is the quality change, and T is the method parameter, called temperature which regulates the probability to accept solutions with higher objective value (cost). Generally speaking, the search starts with a high temperature. Then according to the cooling schedule, the temperature decreases gradually towards the end of the search process. One way of reducing the temperature is to apply the geometric cooling schedule: $T_{i+1} = T_i.\beta$, where β can be empirically tuned for a particular problem domain [45]. Algorithm 3 shows the basic outline of simulated annealing. Due to its relatively easy to code and the ability to generate 'good' solutions, many papers have been published presenting simulated annealing applied to several problem domains, like school timetabling [44] and examination timetabling [46] problems. The disadvantage of this method is that *repeatedly annealing* is a slow process, especially if the objective function value is expensive to compute. Simulated annealing can be used as a move acceptance method within the selection hyper-heuristics [23].

Algorithm 3: Pseudocode of simulated annealing

```
Let S represent the candidate solution, and T the temperature;
Let S_best represent the best solution;
S ← CreateInitialSolution(); // generate random solution
S_best ← S;
T ← T_initial;
repeat
    Generate(S'); // generate neighbour solution S' adjacent to S
    S_best ← maintainBestSolution(S');
    if S' isBetterThan S then
        S ← S';
    end
    else
        p ← e^(−Δ/T);
        if p < UniformRandom[0, 1] then
            S ← S';
        end
    end
    Update(T); // update temperature according to cooling schedule
until TerminationCriterionSatisfied();
return S_best;
```

2.1.4 Great Deluge

The Great Deluge (GD) algorithm was first introduced by Dueck [47]. GD is based on a stochastic framework which allows improving moves by default. Non-improving moves are accepted if the objective value of the candidate solution is better than an expected objective value, named as water level at each step. The water level gets updated according to the 'rain speed' parameter. Dueck [47] argued that if the rain speed value is chosen to be very small then the algorithm produces a high quality solution after a long time. The objective value of the first generated candidate solution can be used as the initial level in GD. The pseudocode of the great deluge is given in Algorithm 4. Dueck [47] proposed two variants of GD named Threshold Accepting (TA) and Record-to-Record Travel (RRT) methods. The idea of TA is based on that any new solution, which is not much worse than the old solution, is accepted. While in RRT, any new solution which is not much worse than the best solution recorded, is accepted. Great deluge can be utilised as a move acceptance method within the selection hyper-heuristics [14].

Algorithm 4: Pseudocode of great deluge

```
Let S represent the candidate solution, B the rain speed and τ the water level;
Let S_best represent the best solution;
S ← CreateInitialSolution(); // generate random solution
S_best ← S;
f_0 ← Evaluate(S); // calculate initial objective function value
τ ← f_0; // initial level
repeat
    Generate(S'); // generate neighbour solution S' adjacent to S
    S_best ← maintainBestSolution(S');
    if Evaluate(S') isBetterThan Evaluate(S) then
        S ← S';
    end
    else
        if Evaluate(S') isBetterThan τ then
            S ← S';
        end
    end
    Update(τ); // update the water level τ = τ ± B
until TerminationCriterionSatisfied();
return S_best;
```

2.1.5 Late Acceptance

The late acceptance hill climbing was recently introduced as a meta-heuristic strategy [48]. Most of the hill climbing approaches modify the current solution and guarantee an equal quality or improved new solution at a given step. The late acceptance hill climbing guarantees an equal quality or improved new solution with respect to a solution which was obtained fixed number of steps before. Algorithm 5 provides the pseudocode of this approach. Late acceptance hill climbing requires implementation of a queue of size L which maintains the history of objective function values of L consecutive visited states for a given problem. At each iteration, algorithm inserts the solution into the beginning of the array and removes the last solution from the end. The size of the queue L is the only parameter of the approach, which reflects the simplicity of the strategy. The late acceptance can be employed as a move acceptance method within the selection hyper-heuristics [16]. Jackson et al. [49] reported that the parameter of the method can influence the performance. If L is too short, the search may quickly converge on a local optimum, if L is too long the search may stagnate.

Algorithm 5: Pseudocode of the late acceptance hill climbing

```
1  Let S represent the candidate solution;
2  Let S_best represent the best solution;
3  S ← CreateInitialSolution(); // generate random solution
4  S_best ← S;
5  f_0 ← Evaluate(S); // calculate initial objective function value
6  for i ← 0, L − 1 do
7  |  f(i) ← f_0;
8  end
9  i ← 0;
10 repeat
11 |  Generate(S′); // generate neighbour solution S′ adjacent to S
12 |  f′ ← Evaluate(S′); // calculate objective function value
13 |  S_best ← maintainBestSolution(S′);
14 |  c ← i mod L;
15 |  if f′betterThanf(c) then
16 |  |  S ← S′;
17 |  end
18 |  f(c) ← Evaluate(S); // include objective value in the list
19 |  i ← i + 1;
20 until TerminationCriterionSatisfied();
21 return S_best;
```

2.1.6 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a class of population (multi-point) based search methodologies, inspired from the Darwinian theory of evolution. A genetic algorithm (GA) meta-heuristic is a subclass of EAs combining principles of natural evolution and genetics for problem solving [37]. A pool of candidate solutions (individuals) for a given problem is evolved to obtain a high quality solution at the end. Mate/parent selection, recombination (crossover), mutation and replacement are the main components of an evolutionary algorithm. A simple GA is outlined in Algorithm 6.

The candidate solutions, referred to as individuals or chromosomes, go through an evolutionary cycle (each referred to as generation) passing through those phases. First, parent individuals are selected from the population, often favouring the high quality solutions, then they are recombined, producing new individuals which are exposed to mutation perturbing those new solutions further. The usefulness of recombination is still under debate in the research community [50, 51]. The replacement operator decides which individuals will survive to the next generation. A memetic algorithm (MA) hybridises a genetic algorithm with local search which is commonly applied after mutation

Algorithm 6: Pseudocode of the genetic algorithm

```
1 CreateInitialSolutions(); // create initial population of solutions
2 repeat
3 |  Evaluate(); // calculate fitness of each solution in the population
4 |  SelectParents(); // select solutions from the population to breed
5 |  Crossover(); // apply crossover operator with a given probability
6 |  Mutate(); // apply mutation operator with a given probability
7 |  ReplaceSolutions(); // generate new population of solutions
8 until TerminationCriterionSatisfied();
```

on the new individuals [52, 53]. A 'meme' may denote a local search method capable of local learning. A memetic algorithm is often tailored for the problem dealt with [54]. Many improvements for MAs have been suggested, for example the population sizing [55] and interleaved mode of operation [56]. MAs have been successfully applied to many different problems ranging from generalised travelling salesman [57] to nurse rostering [58].

2.2 Selection Hyper-heuristics

Selection hyper-heuristics were initially defined as "heuristics to choose heuristics" in [12]. They are capable of selecting and applying an appropriate heuristic given a set of low level heuristics for a problem instance [3].

Özcan et al. [8] identified two successive stages that are common to most of the single point based search hyper-heuristics (that is, without the use of populations of solutions) influencing their performance: *heuristic selection* and *move acceptance*, in attempting to improve a randomly created solution. An initially generated solution is iteratively improved passing through these stages. At each iteration, a candidate new solution is produced by selecting and applying a heuristic (neighbourhood operator) from a set of low level heuristics; A 'move acceptance' component then decides whether or not the candidate should replace the incumbent solution. This cycle continues until a termination criterion is satisfied as illustrated in Figure 2.1 [8]. The processes indicated with bold font in Figure 2.1 take place at the hyper-heuristic level, while the rest take place in the problem domain level. Algorithm 7 illustrates a selection hyper-heuristic framework pseudocode performing a single point search.

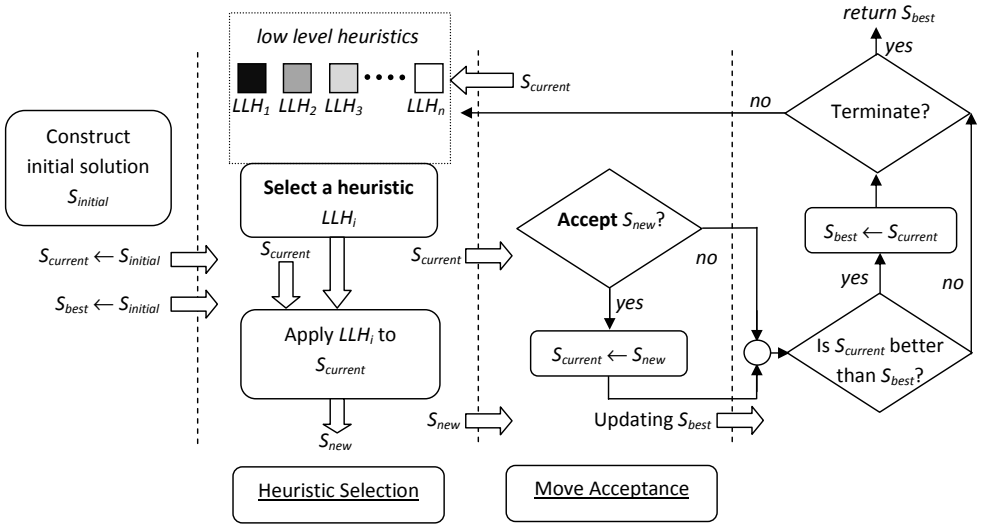


FIGURE 2.1: Illustration of how a single point based selection hyper-heuristic operates

Algorithm 7: Pseudocode of the selection hyper-heuristic framework

Let $S_{current}$ represent the candidate solution;
Let $LLH = \{LLH_1, LLH_2, ..., LLH_n\}$ represent the set of all low level heuristics;
Let S_{best} represent the best solution;
$S_{initial} \leftarrow$ `CreateInitialSolution()`; `// construct initial solution`
$S_{current} \leftarrow S_{initial}$;
$S_{best} \leftarrow S_{initial}$;
repeat
  $LLH_i \leftarrow$ `SelectLowLevelHeuristic`(LLH);
  $S_{new} \leftarrow$ `ApplyHeuristic`$(LLH_i, S_{current})$; `/* generate new solution` S_{new} `by applying` LLH_i `to` $S_{current}$ `*/`
  if `Accept`$(S_{current}, S_{new})$ **then** `// decide whether to accept` S_{new} `or not`
    $S_{current} \leftarrow S_{new}$;
  end
  $S_{best} \leftarrow$ `updateBestSolution`$(S_{current})$;
until `TerminationCriterionSatisfied()`;
return S_{best};

Most of the selection hyper-heuristics in the literature feature a logical boundary filter, referred to as *domain barrier* which disallows any domain specific information to be passed from the problem domain layer to the hyper-heuristic layer during the search process [2, 12]. Hence, selection hyper-heuristics, once implemented, are reusable general methods, applicable to different unseen instances from a specific domain as well as different domains. Even the heuristic selection and move acceptance components of hyper-heuristics can be reused without requiring any change.

2.2.1 Classification of Selection Hyper-heuristics

Depending on the nature of the low level heuristics, two categories of selection hyper-heuristics can be defined, namely, selection perturbative hyper-heuristics and selection constructive hyper-heuristics [2]. Selection perturbative hyper-heuristics process complete solutions, while selection constructive hyper-heuristics process partial solutions.

Selection hyper-heuristics can be further distinguished by their feedback mechanisms and they could either be with online learning, offline learning or no learning at all [13]. The online learning selection hyper-heuristic is the one that learns from feedback during the search process, whereas the offline learning selection hyper-heuristic learns before the actual search starts.

2.2.2 Selection Methods

Cowling et al. [12] presented a variety of selection hyper-heuristics embedding simple heuristic selection methods. They investigated the performance of *Simple Random* (SR), *Random Descent (Gradient)* (RD(G)), *Random Permutation* (RP), *Random Permutation Descent (Gradient)* (RPD(G)), *Greedy* (GR) and a more sophisticated one, namely *Choice Function* (CF) heuristic selection mechanisms. Two acceptance criteria were used to combine with the aforementioned heuristic selection mechanisms. The *All Moves* (AM) acceptance criterion accepts all the generated solutions, while *Only Improving* (OI) accepts only better quality solutions [12]. *Improving or Equal* (IE) accepts non-worsening moves.

Simple random selects a low level heuristic randomly based on a uniform probability distribution at each step, while random descent (gradient) selects a low level heuristic randomly and applies it to the solution in hand repeatedly as long as the solution is improved. If the solution worsens, then another low level heuristic is selected and the same

process is repeated. Random permutation applies a low level heuristic in a randomly generated permutation of all low level heuristics and applies the low level heuristics in the list one after another at each step sequentially. Random permutation descent (gradient) applies the random permutation but proceeds in the same manner as the random descent (gradient) approach without changing the order of heuristics. Greedy applies all low level heuristics to the candidate solution and selects a heuristic, hence a solution, which generates the largest improvement. The new solution could be still worse than the current solution, if all heuristics are performing random perturbation. The authors reported the success of a learning hyper-heuristic; namely, choice function. Greedy also showed some potential. Choice function utilises a mechanism that scores each low level heuristic based on its individual performance, pairwise successive performance and the duration since the last time a heuristic was invoked. The heuristic having the maximum score is chosen and applied to the candidate solution at each step and the relevant information for the chosen heuristic after its application to the current solution are updated. A hyper-heuristic either utilises a learning mechanism or operates without any learning at all [4]. Both GR and CF are online learning mechanisms, since they both learn from their previous experiences by getting and using feedback during the search process. The memory length of the choice function is determined by means of the limits on the score values. A larger range for the score indicates a longer term memory as compared to a lower range. On the other hand, GR has a very short term memory as the feedback (i.e., best candidate solution among all) is used instantaneously and then forgotten in the following step. Following that in different regions of the search space, a different heuristic might operate the best; a good hyper-heuristic is expected to recover the most appropriate heuristic to utilise in a given region as quickly as possible. Therefore, GR allows the search to react faster during the transitions from one region to another in the search space [2]. At the end of the experiments, the authors observed that although greedy heuristic selection was not the best, it delivered a comparable performance to some learning heuristic selection methods. Cowling et al. [12] observed that CF−AM was the most promising hyper-heuristic. The performance strength of CF has been illustrated in the other studies as well [7, 8, 23].

Nareyek [19] uses Reinforcement Learning (RL) that adaptively selects a heuristic from a set of low level heuristics according to the utility values associated with each heuristic. A tabu search based hyper-heuristic ranks the heuristics to determine which heuristic will be selected to apply to the current solution, while the tabu list maintains a list to disallow and avoid the use of low level heuristics performing badly has been suggested by Cowling and Chakhlevitch [59] and Burke et al. [18].

2.2.3 Move Acceptance Methods

The move acceptance criteria in the selection hyper-heuristic frameworks can be classified as *deterministic* or *non-deterministic*. Deterministic move acceptance criteria always return the same decision any time (iteration) they are called with a specific set of input values. Non-deterministic move acceptance criteria depend on the current time or step to make a decision. The latter category can be characterised as *stochastic* if a probabilistic framework is considered while making the decision e.g. simulated annealing; *non-stochastic* if no probabilistic framework is considered while making the decision e.g. great deluge [2, 25]. Existing move acceptance criteria fall in one of the three categories presented in Table 2.1.

TABLE 2.1: Categorisation of some existing move acceptance methods used within the selection hyper-heuristics

	deterministic	*non-deterministic*
stochastic	-	Simulated Annealing variants, EMCQ
non-stochastic	Accept all, Improving or Equal, Only Improving	Great Deluge, Late Acceptance, AILTA

Accepting all moves and some other simple deterministic acceptance methods are described in [12]. There are a number of *deterministic* and *non-deterministic* acceptance methods allowing the acceptance of worsening solutions. Mostly, those methods accept all improving moves, but they differ in how they treat non-improving moves. The non-deterministic naïve move acceptance accepts a worsening solution with a certain probability [60].

Özcan et al. [16] use Late Acceptance [48], which maintains the history of objective values of previously visited solutions in a list of a given size. Ayob and Kendall [15] compared different Monte Carlo based move acceptance criteria which allow the acceptance of non-improving moves using different probability formula. These strategies are similar to the simulated annealing move acceptance (Section 2.1.3) yet without a cooling schedule. Exponential Monte Carlo with Counter (EMCQ) [15] uses the probability of $e^{-\Delta f \times m/Q}$ for accepting non-improving moves, where Δf is the fitness change at a given step, m is the duration of the selected heuristic execution and Q is the number of successive worsening moves. Q is reset whenever there is an improvement. Bilgin et al. [23] tested 36 different hyper-heuristics by pairing up a range of heuristic selection and move acceptance methods over a set of examination timetabling problem instances. A

selection hyper-heuristic using the simulated annealing move acceptance with a linear cooling rate, denoted as SA (Equation 2.1) performed the best when Choice Function is used as the heuristic selection method.

$$p_t = e^{-\frac{\Delta f}{\Delta F(1-\frac{t}{N})}} \tag{2.1}$$

where Δf is the quality change at step t, N is the maximum number of steps, ΔF is an expected range for the maximum quality change in a solution after applying a heuristic.

Kendall and Mohamad [14] experimented with a hyper-heuristic with the SR heuristic selection method and the Great Deluge (GD) (Section 2.1.4) acceptance criterion mechanism. The level is updated at a linear rate towards a final objective value as shown in Equation 2.2.

$$\tau_t = f_0 + \Delta F \times (1 - \frac{t}{N}) \tag{2.2}$$

where τ_t is the threshold level at step t in a minimisation problem, N is the maximum number of steps, ΔF is an expected range for the maximum fitness change and f_0 is the final objective value.

Misir et al. [61] proposed a new threshold move acceptance, named Adaptive Iteration Limited List-based Threshold Acceptance (AILLA). AILLA maintains a list of best l solutions found. Initially, the threshold is set at the level of the previous best found solution. In case no best new solution can be obtained for a number of iterations, a worsening solution is accepted based on the current threshold level. If again no best new solution can be obtained for a number of iterations, then a larger value from the list is used as the new threshold level, and so on.

Four different group decision making strategies are proposed in [25] as a hyper-heuristic move acceptance mechanism: G-AND, G-OR, G-VOT, G-PVO. Each one of these move acceptance mechanisms provides a decision whether a new candidate solution is accepted or not by evaluating the decisions of their member move acceptance mechanisms. Generally speaking, improvements are always accepted and a worsening move subject to the group decision criteria. G-OR and G-AND are biased strategies. G-OR makes an acceptance oriented decision. If the members willing to admit the new solution are in the minority, still, it is accepted. Even if there is a single member that admits the new solution, that member acts as an authority and makes the final decision. On the other hand, G-AND makes a rejection oriented decision. All the member move acceptance

mechanisms must be in agreement so that the new solution gets accepted. Even if the members that reject the new solution are in the minority, it is rejected. G-VOT and G-PVO are based on the majority rule. G-VOT is based on the traditional voting scheme. If the majority of members accept the new solution, it is accepted, otherwise it is rejected. G-AND, G-OR and G-VOT act under certainty, whereas G-PVO is modelled favouring uncertainty to a degree using a probabilistic framework while making the final decision. The probability of acceptance of a new solution dynamically changes proportional to the number of members that vote for acceptance within the group at each step in G-PVO. For example, assuming that there are ten members in the group and six of them accept the new solution at a step, then this solution is accepted by G-PVO with a probability of 0.60. None of the group decision making move acceptance criteria requires an odd number of members, but it is preferable by G-VOT. The proposed group decision making move acceptance criteria can be represented by means of a more general model. In this model, given k move acceptance methods, a move is accepted if the inequality is satisfied by the Equation 2.3, otherwise it is rejected. The contribution of each member move acceptance mechanism towards a final decision for the acceptance can be adjusted through a weight, referred to as *strength* (s_i). Assuming that all s_i values are 1, the method turns out to be G-AND for $\alpha = k$ and G-OR for $\alpha = 0.5$. If $\alpha = k/2$ and all s_i values are 1, then the method becomes G-VOT. If $\alpha = k \times r$, where r is a uniform random number in [0,1] and all s_i values are $1/k$, then the method becomes G-PVO.

$$\sum_{i=1}^{k} s_i \times D(M_i) \geq \alpha \tag{2.3}$$

where M_i denotes the i^{th} group member (a move acceptance mechanism), $D(x)$ returns 1, if the strategy x accepts the new solution and 0, otherwise, s_i is the strength of the decision made by the i^{th} member move acceptance mechanism and α denotes a threshold value.

2.2.4 Selection Hyper-heuristic Frameworks

Özcan et al. [13] demonstrate and compare between four selection hyper-heuristic frameworks discriminating between *mutational* and *hill climbing* low level heuristics: F_A, F_B, F_C and F_D. A hyper-heuristic framework without differentiating the low level heuristics (F_A) is presented in [18]. The mutational low level heuristics perturb a given solution mostly at random and act as a diversification component which enables the search process to explore the other regions potentially leading to high quality solutions. The

hill climbing heuristics always produce non-worsening solutions. F_B selects a low level heuristic from a set of mutational and hill climbing low level heuristics. If the selected heuristic is mutational, then a hill climber is applied further before the move acceptance decides on accepting or rejecting the new solution. F_C selects one of the mutational heuristics followed by a hill climber before the acceptance decision is made. F_D separates the mutational from the hill climbers. First a mutational heuristic is selected and applied, the new solution is passed to the acceptance method and an intermediate solution is generated, which is passed to the hill climbers' phase where a hill climber is selected and applied to the intermediate solution. A separate decision is then made to accept or reject the new solution. The experimental results on a set of benchmark functions showed that the best performing hyper-heuristic framework applies a predetermined hill climber right after a mutational heuristic similar to the process in iterated local search [13].

2.2.5 Other Trends in Selection Hyper-heuristics

Other recent trends in the domain of selection hyper-heuristics include the application of selection hyper-heuristics in dynamic environments [62, 63], and multi-objective optimisation problems [64, 65]. Selection hyper-heuristics can also be employed to hybridise search. As an example, the low level heuristics in [66] were different search method algorithms such as breadth-first search, depth-first search, best-first search, hill-climbing and A* search algorithm. Maashi et al. [65] also employed selection hyper-heuristics to manage a set of multi-objective evolutionary algorithms. A further development in the selection hyper-heuristic domain is the use of a unifying 'mathematical' formulation as a blackboard architecture for designing hyper-heuristics [67]. Also, a growing number of studies focused on running hyper-heuristics in a distributed setting in order to facilitate higher performance. Ouelhadj et al. [68] provided an agent-based multi-level search framework for the asynchronous cooperation of hyper-heuristics.

The idea of applying different hyper-heuristics at different stages has not been well studied previously. For instance, Chakhlevitch and Cowling [69] proposed a multi-stage hyper-heuristic by applying one of two hyper-heuristics at each stage. The first hyper-heuristic employs a Greedy approach which is used to reduce the number of low level heuristics. In the following stage, a simple random hyper-heuristic accepting non-worsening moves is used. The authors reported that using both greedy and tabu search in combination with the aim of linearly reducing the number of the best performing low

level heuristics, is promising. Lehre and Özcan [24] conducted a theoretical study using a selection hyper-heuristic on a benchmark function showing that an improved run-time complexity can be obtained by mixing simple move acceptance criteria rather than using each move acceptance method on its own. In that study, random choice is used as a heuristic selection and the algorithmic framework could be viewed as a multi-stage framework in which two hyper-heuristics with different move acceptance are employed. In this study, the multi-stage hyper-heuristic framework is investigated.

More on hyper-heuristics including the descriptions of more elaborate selection hyper-heuristic components as well as other types of hyper-heuristics can be found in [2, 3, 6, 8, 70].

2.3 HyFlex Problem Domains and Cross-Domain Heuristic Search Challenge

There are tools like Hyperion [71] and HyFlex[1] [33] which are available for rapid development and research of hyper-heuristics or meta-heuristics. Hyperion [71] provides a general recursive framework for the development of hyper-heuristics (or meta-heuristics), supporting the selection hyper-heuristic frameworks provided in [8]. HyFlex provides an excellent controlled environment for evaluating a new hyper-heuristic approach and comparing its performance to the other approaches. It provides an object-oriented reusable hyper-heuristic (meta-heuristic) framework interface written in Java, having the implementation of six problem domains each with different instances and a set of relevant low level heuristics. HyFlex currently provides implementation of six minimisation problem domains: boolean satisfiability (SAT), one-dimensional bin-packing (BP), personnel scheduling (PS), permutation flow-shop (PFS), travelling salesman problem (TSP) and vehicle routing problem (VRP). The software package includes a set of low level heuristics and a number of instances associated with each domain. There is a growing number of work on selection hyper-heuristics which have been designed and tested using HyFlex. HyFlex strictly imposes the domain barrier and does not give users any access to the problem domain dependent information. In HyFlex, the low level heuristics are perturbative heuristics processing and returning complete solutions after their application. Heuristics are categorised as *mutational*, which modify a solution in some way with no guarantee of improvement, *ruin and re-create* heuristic, which destruct a given complete

[1] `http://www.hyflex.org/`

solution generating a partial solution and then reconstruct a complete solution, *hill climbing*, which perform local search returning a solution which has the same or better quality of the input solution, and *crossover*, which create a new solution by combining some parts from two given solutions.

Burke et al. [60] 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[2] in 2011. The Formula One points scoring system is used for comparing the performance of hyper-heuristics. The top hyper-heuristic receives 10 points, the second gets 8 and then 6, 5, 4, 3, 2, 1, respectively. The remaining approaches get zero points. These points are accumulated as a score for a hyper-heuristic over all instances. A run terminates after 600 seconds or equivalent to 10 minutes as the competition requires. The equivalent value can be obtained using the benchmarking tool provided at the competition website.

Prior to the actual competition of CHeSC 2011, the organisers arranged a mock competition using eight hyper-heuristics (HH1-HH8) across a subset of four CHeSC problem domains. A single run is performed across 10 instances of boolean satisfiability, one-dimensional bin-packing, personnel scheduling and permutation flow-shop problem domains in the mock competition. The maximum overall score that a hyper-heuristic can achieve is 400. The results of this mock competition were provided for the competitors to form a baseline and assess the performance of their algorithms. The mock competition hyper-heuristics were designed based on the previously proposed well known techniques from the literature. The description of the mock hyper-heuristics was not provided on the competition website, but it is reported in [60] that the iterated local search which applies a sequence of heuristics in a predefined order has the best performance. This framework is based on the most successful hyper-heuristic framework reported to perform the best in [8].

In CHeSC 2011, the competing hyper-heuristics are run for thirty one trials on the reference machine and the median result (16^{th}) is used for comparison of the approaches based on the Formula One points scoring system. The 20 submitted hyper-heuristics competed over thirty problem instances, five coming from each of the six problem domains: boolean satisfiability, one dimensional bin packing, permutation flow shop, personnel scheduling, travelling salesman problem and vehicle routing problem. Two instances of the mock problem domains were hidden. The maximum overall score that a hyper-heuristic could

[2]CHeSC 2011 website: `http://www.asap.cs.nott.ac.uk/chesc2011/`

achieve is 300. The results of the twenty participants in the competition (Table 2.2) along with the description of their algorithms were provided in the website of the competition.

Table 2.2: The CHeSC 2011 competing approaches

Rank	Method label	Score	Reference	Rank	Method label	Score	Reference
1	AdapHH	181.00	[27]	11	ACO-HH	39.00	[72]
2	VNS-TW	134.00	[73]	12	GenHive	36.50	[74]
3	ML	131.50	[75]	13	DynILS	27.00	[76]
4	PHUNTER	93.25	[77]	14	SA-ILS	24.25	–
5	EPH	89.75	[78]	15	XCJ	22.50	–
6	HAHA	75.75	[79]	16	AVEG-Nep	21.00	[80]
7	NAHH	75.00	[81]	17	GISS	16.75	[82]
8	ISEA	71.00	[83]	18	SelfSearch	7.00	[84]
9	KSATS-HH	66.50	[85]	19	MCHH-S	4.75	[86]
10	HAEA	53.50	[87]	20	Ant-Q	0.00	[88]

Soon after the competition, CHeSC 2011 became a benchmark for evaluating the performance and generality level of a selection hyper-heuristic. There is a growing number of studies evaluating the performances of new selection hyper-heuristics on the CHeSC 2011 benchmark. Drake et al. [89] tested a variant of choice function hyper-heuristic over the CHeSC 2011 benchmark. An adaptive neighbourhood iterated local search is proposed and applied on HyFlex problem domains [90, 91]. In [92], an iterated local search method is tested on HyFlex problem domains. Jackson et al. [49] evaluated variants of *late acceptance* based selection hyper-heuristics on the CHeSC 2011 benchmark and points out the best configuration for the late acceptance strategy which accepts the current solution if its quality is better than the quality of a solution obtained from a certain number of iterations ago.

2.4 High School Timetabling Problem

The educational timetabling problems, such as university course timetabling, examination timetabling and high school timetabling, are well known real-world constraint optimisation problems which have been of interest to researchers as well as practitioners across operational research, computer science and artificial intelligence since 1960s [93]. Due to the intrinsic difficulty and NP-hard nature of the problems [30, 94], the traditional exact approaches might fail to find a solution in a given time even for moderately sized versions of the problem.

High school timetabling is the focus of this study [95], which requires a search for the best course schedule and the best allocation of limited resources in an educational institution subject to a set of constraints. High school timetabling is different from university course timetabling. The main difference is that the timetable for a student is more packed in high schools and students are fully occupied throughout a day. Consequently, the shared resources are more loaded. There are two basic constraint types in timetabling problems: $hard$[3], and $soft$. The hard constraints require absolute compliance, whereas the soft constraints characterise preferences. Types (and categories) of the constraints, number of the constraints, courses to be scheduled and the resources in a problem might influence its difficulty level. Timetabling problems have been studied by many researchers [96–99].

Due to the nature of timetabling problems (e.g., unstructured search space, immense size of the search landscape, constraints etc.), meta-heuristics are preferred in most of the previous studies. There are a variety of real world timetabling problems exhibiting various characteristics from different countries, and many different meta-heuristic approaches have been proposed for a particular problem in hand, ranging from single point based search methods, including simulated annealing (SA) and tabu search (TS) to population based methods, such as, evolutionary algorithms including genetic algorithm (GA), memetic algorithm (MA) and ant colony optimisation. These methods are frequently preferred for solving different types of timetabling problems across different institutions. The hybrid population based evolutionary algorithms for high school course timetabling have been growing since the 1990s [98, 100, 101]. Abramson [102] utilised simulated annealing for course timetabling and proposed a parallel algorithm for solving some randomly generated problem instances and some Australian high school data. Colorni et al. [103] evaluated various meta-heuristics based on genetic algorithm, simulated annealing and tabu search on some Italian high school data. They observed that memetic algorithm hybridising genetic algorithm with local search performed better. Hertz [104] employed tabu search for teacher-course assignment using data from a Yugoslavian school. In [105], tabu search is employed for obtaining the course schedules on high schools data. Bello et al. [106] tested a tabu search approach on some instances from Brazilian high school timetabling problems. Erben and Keppler [98] generated a weekly timetable for a heavily constraint problem instance using genetic algorithms with smart operators. They used binary encoding as a representation scheme. Schaerf

[3]In most of the cases, a *feasible* solution which satisfies the hard constraints is sought. In the ITC 2011 competition, such constraints are not strictly hard but are simply much more heavily penalised than the 'soft' constraints.

[107] proposed an interactive interface for timetabling and tested a tabu search based approach which interleaves different types of moves on some instances from the Italian high schools. The approach generated schedules that are of better quality than the manually created ones. Beligiannis et al. [108] presented an evolutionary algorithm which employs no crossover and multiple mutation operators. A comparison to the previously proposed approaches of column generation and constraint programming on a Greek school course timetabling problem revealed the success of the approach. Jacobsen et al. [109] presented a tabu search algorithm for solving a timetabling problem at German secondary schools of Gymnasium type and compared its performance to a constraint programming approach. The results showed that they have a similar performance based on the feasible solutions obtained for the given instances. Filho et al. [110] formulated a timetabling problem as a clustering problem and applied a constructive genetic algorithm for solving timetabling problems of public schools in Brazil. Wilke et al. [111] proposed a hybrid genetic algorithm using multiple genetic operators and a parameter configuration strategy that randomly chooses from different options during the search process whenever the algorithm detects that no improvement can be made. The results showed that the proposed hybrid approach performed better than the traditional genetic algorithm on a large German high school problem instance. Raghavjee and Pillay [112] compared the performance of a genetic algorithm, neural network, simulated annealing, tabu search and greedy search on the problem instances provided by Abramson and Dang [113]. The experimental results showed that genetic algorithm delivered either a better or similar performance to the previously proposed methods. Raghavjee and Pillay [114] described a hybrid evolutionary algorithm with no crossover using a hill climber for solving a South African high school course timetabling problem along with a primary school timetabling problem. Kannan et al. [115] applied a graph theoretic approach to a problem from the New York City public school system, which decomposes a given instance and applies randomised heuristics. Alkan and Özcan [116] hybridised a violation directed hierarchical hill climbing method (VDHC) using constraint oriented neighbourhood heuristics with genetic algorithms for solving the university course timetabling problem. Similarly, the constraint oriented neighbourhood heuristics were found to be effective when used as a part of a hybrid framework in [56] for solving a variant of a high school course timetabling problem. In [117], a greedy randomised adaptive search procedure (GRASP) heuristic is applied Brazilians high schools data. Pillay [118] implemented an evolutionary algorithm based hyper-heuristic selection method. The study revealed that the incorporation of local search heuristics with mutation and crossover operators improves the performance. The approach outperforms the other methods applied to the same problem. Özcan et al.

[56] introduced a variant of a high school timetabling problem from Turkey and proposed a genetic algorithm hybridised with hill climbing which interleaves the proposed algorithm with constructive methods while exploiting the underlying hierarchical structure of a given problem. In this study, rather than attempting to develop tailored solutions similar to the approaches provided above, we prefer the use of general hyper-heuristic, and particularly, the developed multi-stage hyper-heuristic framework to solve the problem. Pillay [119] surveyed the use of hyper-heuristic approaches in solving timetabling problems.

Due to the variety of existing high school timetabling problems and sometimes lack of algorithmic details, it is not trivial to implement and compare the performance of different approaches. The International Timetabling Competitions have been organised with the goal of encouraging researchers and practitioners to design solution methods for real world problems incorporating all real world complexities into their models and form real world benchmark for the timetabling community. The state-of-the-art methods for a given domain has always been of interest for researchers as well as practitioners, which has been the case for timetabling as well. The Third International Timetabling Competition (ITC 2011)[4] was recently organised after ITC 2002 (http://www.idsia.ch/Files/ttcomp2002/) and ITC 2007 [120] which were on educational timetabling, mainly focusing on university course and examination timetabling.

The challenge on high school timetabling has become increasingly highlighted when a group of researchers run the Third International Timetabling Competition (ITC 2011) in 2011-2012 [121], with the goal of raising the profile of automated high school timetabling. The ITC 2011 was run by the Centre for Telematics and Information Technology at the University of Twente in the Netherlands, aiming to drive a new era of research of automated high school timetabling. The problem instances, obtained from different countries across the world used in this competition became a benchmark for further research in the field. Some of these instances are used to test the methods in the literature. Briefly, the ITC 2011 problem instances contain 15 types of constraints and a candidate solution is evaluated in terms of two components: feasibility and preferences. The evaluation function computes the weighted hard and soft constraint violations for a given solution as *infeasibility* and *objective* values, respectively. For the comparison of algorithms, a solution is considered to be better than another if it has a smaller infeasibility value, or an equal infeasibility value and a smaller objective value.

[4]ITC 2011 website: `http://www.utwente.nl/ctit/hstt/`

Out of 17 registered participants to the competition, only 5 teams submitted solutions. The reason is unknown, but could be due to the large number of imposed constraints which makes the problem hard to handle in practice [122]. The competition consisted of three rounds. In the first round, competitors were invited to submit solutions to all public instances with the goal of finding the best approach that improves upon the best known solutions from the literature for each instance. No restrictions were placed on the time limit or how the solutions could be obtained. In the second round, solvers were compared under uniform conditions. Solvers were allowed to use only freely available software libraries, and a time limit was imposed as 1000 nominal seconds based on the organisers' computer. For each of the hidden instances, ten runs with different random seeds were conducted considering submission of stochastic algorithms. The solutions obtained from each run for each instance were ranked and then averaged to determine the winner. Five finalists have been selected up to check the solvers with hidden instances: HySST, GOAL, HFT, Lectio and VAGOS. Four solvers, each identified by the name of the designing team were submitted to Round 2 of the ITC 2011 competition [122]. HySST [43] applied a multi-stage hyper-heuristic managing a set of mutational heuristics and two hill climbers. This selection hyper-heuristic incorporates random choice for the heuristic selection and an adaptive threshold move acceptance method. HFT [123] used an evolutionary algorithm as a solution method. Lectio [124] employed an approach based on adaptive large neighbourhood search. GOAL [125] combined iterated local search based on multiple neighbourhood operators with simulated annealing, which turned out to be the winner of the second round of the competition.

In the third round, the hidden instances were published and the competitors were invited to submit the best solutions that they can achieve by any algorithm. The same ranking strategy as the second round was used during this round to determine the winner. As for Round 1, no restrictions are placed on how the solutions could be obtained.

For a recent survey of HSTP see [95, 119, 126].

2.5 Multi-mode Resource-constrained Multi-project Scheduling Problem

Project scheduling has been of interest to academics as well as practitioners. The broad aim is to schedule a set of different and partially interacting projects. Each project

consists of a set of activities. The activities must respect a set of (hard) precedence constraints and project release times. Also the activities use resources and the appropriate resource limits are also hard constraints. There are a variety of project scheduling problems and there are many relevant surveys on this topic in the literature [127–134]. The simplest version of the problem is the resource-constrained project scheduling problem (RCPSP). In RCPSP, activities (jobs) are respecting given precedence relationships and often performed simultaneously in only one way, called a single mode. The allocation of scarce resources among the different project activities to achieve the optimisation of an objective function is an important consideration for the project planners [128, 135]. In multi-mode RCPSP which was introduced by Elmaghraby [136], each activity actually can be performed using any one of a set of 'modes'. The mode determines the set of resources used by the activity and the duration of the activity (though note that the modes do not affect the set of precedences). A solution consists of an assignment of mode and starting time to every activity and that satisfies all the precedence and resource constraints. The multi-mode resource-constrained multi-project scheduling problem (MRCMPSP) is a general class of RCPSP. In MRCMPSP, activities of multiple projects are scheduled, while taking into account given precedence relationships and availability of limited resources. The RCPSP belongs to the class of NP-hard problems identified by Blazewicz et al. [137], hence the general form MRCMPSP is also NP-hard. There are basically two different ways to distinguish resources. Firstly, they can be either renewable or non-renewable [138]. Renewable resources are ones that are available again at their full capacity whenever current activities stop using them, for example, they could be some machine. Non-renewable ones disappear on usage; an example could be fuel where one can take any amount of fuel, but only until the tank is empty. In particular the mode can affect the usage of non-renewable resources. The second distinction between resources is that of 'local' resources that are associated with one of the projects, and 'global resources' that are shared between different projects.

There have been a growing number of studies concerning the RCPS and RCMPS problems. The traditional optimisation techniques such as integer programming [139, 140] and zero-one (0-1) linear programing [141], showed success in solving small sizes of the multi-project scheduling problems. However, such methods cannot be employed when the number of projects and activities increase and when several resources are considered. Instead, researchers have made several efforts to develop efficient heuristic and meta-heuristic methods to generate the schedules. One of the meta-heuristic methods used for the scheduling problems is genetic algorithm (GA) [142–144]. Up to 90% of

projects [145] in Research and Development (R&D) organisations [146] and large construction companies [147] are in the context of MRCMPSP. Tseng [135] employed a parallel scheduling and genetic algorithms for the MRCMPSP. Can and Ulusoy [148] developed a two-stage decomposition approach and genetic algorithm to solve the MRCMPSP. Xu and Feng [149] utilised a hybrid particle swarm optimisation method and tested the developed approach in a large scale hydropower construction project. Ju and Chen [150] developed a design structure matrix to model the MRCMPSP problem and then applied a modified artificial immune network algorithm (aiNet) to solve the problem. The approach delivers the best results when compared to genetic algorithm, simulated annealing and ant colony optimisation methods.

Recently, a challenge in the context of the 6th Multidisciplinary International Scheduling Conference (MISTA 2013[5]), is run at which competitors are expected to submit solvers to automate the scheduling of multi-mode resource-constrained multi-project problem. The participants of MISTA 2013 tackled 30 instances of the MRCMPSP, produced by combining several multi-mode resource-constrained project scheduling problem (MRCPSP) instances. The MRCPSP instances are generated by the standard project generator ProGen [151]. The instances are defined by a standard data format based on PSPLIB[6] (project scheduling problem library) data format. In the MISTA 2013 Challenge, there were no global non-renewable resources, and so the only interaction between projects is from the global renewable resource(s). Each project p in a given solution in MISTA 2013 has an associated makespan MS_p which is the time from it being released to the time the last activity is completed. The primary objective is to minimise the "Total Project Delay" (TPD), which (up to constant terms) is the sum of the difference between the critical path durations CPD_p and the makespans MS_p for each project p. CPD_p is the shortest MS_p resource unconstrained duration of project p. The tie-breaking secondary objective is to minimise the overall "Total Makespan" (TMS), which (up to a constant term) is the finishing time of the last activity.

During the MISTA challenge, the first set of 10 instances (set-A) was released during the qualification phase and the participants were invited to submit the solvers and solutions to those instances. The organisers compared the solvers under uniform conditions where the imposed time limit was five minutes of multi-threaded execution per instance on the organisers' computer. A set of qualified teams were determined at the end of the qualification phase. Subsequent to the qualification phase, a second set of 10 instances (set-B) were published. Again, the finalists were invited to submit the solvers

[5]MISTA 2013 website: `http://allserv.kahosl.be/mista2013challenge/`
[6]PSPLIB benchmark: `http://129.187.106.231/psplib/`

and solutions to those instances. The organisers ran the solvers on a set of instances from set-B and another set of hidden (unpublished) instances (set-X) to decide on the winner of the challenge. The hidden instances (set-X) were published after the challenge ended. Ten runs with different random seeds each for 300 seconds were conducted. The final objective values obtained from each run for each instance were ranked and then averaged to determine the winner of the MISTA 2013. The team ranking results and the number of the best solutions, out of the twenty instances, obtained are provided in Table 2.3 and in the competition website. The winner approach is described in the thesis. Briefly, it consists of a two-phase construct-and-improve method working on the sequence in which activities are given to a schedule constructor. The construction of an initial activity sequence is done by a (novel) hybrid of MCTS and partitioning of the projects. The improvement phase uses a large number of neighbourhood moves, in a multi-threaded fashion, and controlled by a mix of ideas from meta-heuristics, memetic algorithms, and multi-stage hyper-heuristics. The second approach [152] is based on variable neighbourhood search and iterated local search. The third team proposed an integer programming approach [153] and soon after the competition, they improved the approach by hybridising a local search method [154].

TABLE 2.3: The MISTA 2013 competing approaches

Rank	Team ID	Score	#best solutions	Reference
1	11	1.10	17	[155]
2	8	2.55	1	[152]
3	1	3.05	2	[153]
4	20	3.60	0	[156]
5	13	6.75	0	[157]
6	15	6.75	0	[158]
7	17	6.75	0	[159]
8	14	6.85	0	[160]
9	21	7.60	0	[161]

2.6 Constructing Magic Squares

A square matrix of distinct numbers $(1, ..., n^2)$ in which every row, column and both diagonals has the same total is referred to as a magic square. Constructing a magic square of a given order is considered as a computationally difficult permutation problem, particularly when additional constraints are imposed. The history of magic squares

dates back to 2200 B.C. [32]. An unusual numerical pattern found by Emperor Yu on a tortoise's shell was the oldest known magic square. The Emperor decided to call this unique diagram "Lo-Shu" (Figure 2.2).

FIGURE 2.2: The Loh-Shu tortoise and the magic square [1]

The Chinese have used the magic squares in the interpretation of philosophy, human behaviour, natural phenomena and other areas of study; and interestingly, some of the porcelain plates in some private collections and museums in China were decorated with magic squares. It is thought that the magic squares were transmitted to the Arabs from the Chinese, probably through India. Magic squares were then introduced to Europe, then journeyed to Japan. Magic squares in India were used in applications other than only in the traditional mathematical context. A sequence of naïve rules to construct magic squares were made by Islamic mathematicians. The seventeenth century witnessed a serious consideration to the study of magic squares when Antoine de la Loubere, a French aristocrat, studied the theory behind the construction of magic squares. The extension of magic squares to 3-dimension was brought by Adamas Kochansky in 1686. Recently, the magic squares attracted researchers and applied in statistics, combinatorial mathematics, artificial intelligence, graph theory, industrial arts, experiment designs, location analytics, electronic circuits and more [1, 32].

Kraitchik [31] provided an exact solver to construct the magic squares. A magic square of an odd order can be generated using the Siamese method (also known as De la Loubére's method). An odd order magic square is of the form $n = 2m + 1$, where m is an integer greater than 0. In the Siamese method, the number 1 is written in the middle of the first row. The remaining numbers are placed in ascending order as an upward diagonal to the empty right square cells. In case the cell is already filled, then the cell below the previous number is used to place the number. A magic square of a doubly even order can be generated using cross method. A doubly even order magic square is of the form $n = 4m$, where m is an integer greater than 0. The idea is to draw a cross through every 4x4 sub-square and then fill out all the square cells with all numbers in ascending order

from the top left of the square to the bottom right. Then, each number, a_{ij}, along a diagonal of the cross is replaced by $(n^2 + 1) - a_{ij}$. Finally, a magic square of a singly even order can be generated using the "LUX" method which has been proposed by J. H. Conway. A singly even order magic square is of the form $n = 4m + 2$, where m is an integer greater than 0. The method starts by creating $m + 1$ rows of L, then 1 row of U followed by $m - 1$ rows of X. Then replacing the U in the centre with the L above it. The resulted letters form a square of an odd order $2m + 1$. Constructing the singly even order magic square is done by using the Siamese method and filling out each set of square cells surrounding a letter sequentially according to the shape of the letter. Weisstein [162] reported other methods for generating magic squares.

Although there is at most only one distinct magic square of order less than 4, the number of magic squares of order 4 is 880 as has been known since the seventeenth century. The exact number of distinct magic squares of order 5 is 275,305,224 [32]. Researchers claimed that determining the number of distinct magic squares of order of 6 and more is a hard unsolved computationally problem [163, 164]. Pinn and Wieczerkowski [164] used a Monte Carlo method to predict the number of magic squares of order 6 and their estimate was $(0.17745 \pm 0.00016) \times 10^{20}$.

The exact deterministic methods for constructing magic squares similar to the ones presented above can only produce a single magic square of a given order. Such methods may fail when some constraints are imposed. Xie and Kang [32] proposed a stochastic constructor method based on an improved evolutionary algorithm. The constraint version of the magic squares problem was then the subject of a competition hosted by SolveIT Software[7] with the goal of finding the *quickest* approach. The winner approach emerged among hundreds of competing algorithms as a late acceptance hill climbing algorithm [165] which handles a given instance in two separate ways based on its size. The approach mixes two heuristics with a certain probability for problems larger than a certain size and uses a different algorithm for smaller instances. The winner approach was able to construct the constrained version of the 2600x2600 magic square. Geoffrey Chu developed a solver in which a random square is transformed into the magic square by the iterative heuristic improvement of rows and columns. Chu's solver ranked the second on the competition and it was able to construct the constrained version of a 1000x1000 magic square in one minute. The multi-step iterative local search took the third place on the competition. It was developed by Xiao-Feng Xie and it was able to construct

[7]SolveIT competition website: `http://www.solveitsoftware.com/competition.jsp`

constrained version of a 400x400 magic square in one minute. The detailed descriptions of the top three solvers are available online at `http://www.cs.nott.ac.uk/~yxb/IOC/`.

Kheiri and Özcan [166] extended the framework of the winning approach to enable the use of selection hyper-heuristics for any given constraint version of the magic square problem. They presented a range of effective selection hyper-heuristics combining different heuristic selection methods and acceptance criteria and mixing the same set of perturbative low level heuristics for constructing the constrained version of magic squares. The seven heuristic selection methods {GR, SR, RD, RP, RPD, CF, TABU} are combined with six move acceptance methods {accepting all moves, accepting only improving moves, accepting improving or equal moves, simulated annealing, great deluge, naïve move acceptance} producing a total of 42 selection hyper-heuristics for experimentation. All selection hyper-heuristics are tested with the goal of detecting the quickest one. Greedy based hyper-heuristics and any hyper-heuristic using one of the move acceptance methods in {accepting all moves, accepting only improving moves, accepting improving or equal moves, simulated annealing, great deluge} failed to construct the constraint-version of magic squares within the time limits. The experiments show that hyper-heuristics using the naïve move acceptance method, which accepts a worsening solution with a probability of 0.004%, is the most successful approach. The random permutation based selection hyper-heuristic combined a naïve acceptance method ($RP - NAM$) turns out to be an extremely effective and efficient approach which runs faster than all other hyper-heuristics using different move acceptance methods. It has been observed that learning requires time slowing down a selection hyper-heuristic and so hyper-heuristics with no learning using the naïve acceptance method are more successful than the learning hyper-heuristics regardless of whether the learning occurs within the heuristics selection or move acceptance component. $RP - NAM$ outperforms the best known heuristic approach based on late acceptance for constructing a constrained magic squares.

2.7 Summary

This chapter briefly presented a survey of selection hyper-heuristics; and an overview of the approaches proposed previously for solving the problems dealt with in this work.

Several studies [8, 13, 23] showed that using different combinations of the selection hyper-heuristic methods yields to a different performance. On another studies [24, 68, 69],

it has been suggested to combine several selection hyper-heuristics in a single method. This situation points out a potential modification of the generic selection hyper-heuristic framework. In this thesis, the framework that managing a number of selection hyper-heuristics, referred to as multi-stage hyper-heuristic, is proposed.

The HyFlex problem domains, high school timetabling, multi-mode resource-constrained multi-project scheduling, and construction of magic squares problem domains are used as benchmark case studies for the development of multi-stage hyper-heuristic framework. These problems were subjects of recent competitions, and therefore, we summarised information relevant to the competitions and the methods entered into these competitions to test the level of generality of the proposed multi-stage hyper-heuristics in environments similar to the competitions' environments.

Chapter 3

Problem Domains

The main goal of the hyper-heuristic researches is to raise the level of generality and apply them to a wide range of problem domains without additional effort on the hyper-heuristic side. Therefore, the applicability of the multi-stage hyper-heuristics in this study has been tested on a set of problem domains. There was not any particular consideration taken into account while selecting these problem domains. This chapter describes the studied problem domains including the characteristics of relevant benchmark instances and covers some domain level design details, such as, low level heuristics and solution construction methods. Full descriptions of the problems can be found at the competitions' websites (see Table 3.1) and elsewhere; however, for completeness, we provide a summary of each problem dealt with in this chapter.

3.1 CHeSC 2011: HyFlex Problems

HyFlex currently provides implementation of six minimisation problem domains: boolean satisfiability (SAT), one-dimensional bin-packing (BP), personnel scheduling (PS), permutation flow-shop (PFS), travelling salesman problem (TSP) and vehicle routing problem (VRP). The software package includes a set of low level heuristics (LLHs) and a number of instances associated with each domain. The code for SAT, BP, PS and PFS is released first along with some public instances, while the code and instances for TSP and VRP released after the competition.

TABLE 3.1: Problem domains tested in this study

Problem Domain	Competition: Website
Boolean Satisfiability One-dimensional Bin-packing Personnel Scheduling Permutation Flow-shop Travelling Salesman Vehicle Routing	CHeSC 2011: `http://www.asap.cs.nott.ac.uk/chesc2011/`
High School Timetabling	ITC 2011: `http://www.utwente.nl/ctit/hstt/`
Multi-mode Resource-constrained Multi-project Scheduling	MISTA 2013: `http://allserv.kahosl.be/mista2013challenge/`
Constructing Magic Square	SolveIT 2011: `http://www.solveitsoftware.com/competition.jsp`

Any problem domain developed for HyFlex is required to define a set of low level heuristics (neighbourhood operators) which should be classified as *mutational* (MU), *hill climbing* (HC), *ruin and re-create* (RC) or *crossover* (XO). All heuristics are perturbative. A mutational heuristic makes a random perturbation producing a new solution and this process does not necessarily generate an improvement over the input solution. Local search or hill climbing is often an iterative procedure searching different neighbourhoods starting from a given solution. A ruin and re-create operator produces a partial solution from a given complete solution and then rebuilds a new complete solution. Crossover is a well known operator in evolutionary computation, which takes two solutions and produces a new solution. In general, crossover yields two new solutions and the best new solution is returned in HyFlex.

HyFlex provides utilities to control the behaviour of some low level heuristics to a limited extent. It is possible to increase or decrease the intensity of some mutational and ruin and re-create operations by adjusting a control parameter from 0.0 to 1.0. Changing the value of the intensity parameter could mean changing the range of new values that a variable can take in relation to its current range of values or changing the number of solution variables that will be processed by a heuristic. There is also another similar control parameter for some local search operators for changing the depth of search which relates to the number of hill climbing steps.

The number of the low level heuristics for each heuristic/operator type for each problem domain is presented in Table 3.2. Currently, there are 12 different instances for the first four problem domains and 10 for the last two problem domains. What is left for the researchers and practitioners is to design and implement a general high-level strategy (hyper-heuristic) that intelligently selects and applies at each decision point suitable low level heuristics from the set provided to each instant from the given domain to improve an initially generated solution and to get the minimum objective function value in ten minutes. The nature of each low level heuristic for each HyFlex problem domain is summarised in Table 3.3. OPid is used to denote the id^{th} low level heuristic of type OP. For example, MU0 and MU5 for SAT are the 0^{th} and 5^{th} mutational low level heuristics in the SAT domain.

HyFlex does not provide any annotation for the low level heuristics in a given domain, indicating whether they operate on a given solution with a stochastic or deterministic approach. Although this could be detected with a level of certainty using some initial tests over the set of heuristics, it is assumed that all operators are stochastic.

TABLE 3.2: The number of different types of low level heuristics {mutation (MU), hill climbing (HC), ruin and re-create (RC), crossover (XO)} used in each problem domain

Domain	MU	HC	RC	XO	Total
SAT	5	2	1	2	10
BP	3	2	2	1	8
PS	5	4	2	4	15
PFS	1	5	3	3	12
TSP	5	3	1	4	13
VRP	3	3	2	2	10

TABLE 3.3: The nature of the low level heuristics used in each problem domain. The bold entries for each problem domain mark the last low level heuristic of each type

LLH IDs	LLH0	LLH1	LLH2	LLH3	LLH4	LLH5	LLH6	LLH7
SAT	MU_0	MU_1	MU_2	MU_3	MU_4	$\mathbf{MU_5}$	$\mathbf{RC_0}$	HC_0
BP	MU_0	RC_0	$\mathbf{RC_1}$	MU_1	HC_0	$\mathbf{MU_2}$	$\mathbf{HC_1}$	$\mathbf{XO_0}$
PS	HC_0	HC_1	HC_2	HC_3	$\mathbf{HC_4}$	RC_0	RC_1	$\mathbf{RC_2}$
PFS	MU_0	MU_1	MU_2	MU_3	$\mathbf{MU_4}$	RC_0	$\mathbf{RC_1}$	HC_0
TSP	MU_0	MU_1	MU_2	MU_3	$\mathbf{MU_4}$	$\mathbf{RC_0}$	HC_0	HC_1
VRP	MU_0	MU_1	RC_0	$\mathbf{RC_1}$	HC_0	XO_0	$\mathbf{XO_1}$	$\mathbf{MU_2}$

LLH IDs	LLH8	LLH9	LLH10	LLH11	LLH12	LLH13	LLH14
SAT	$\mathbf{HC_1}$	XO_0	$\mathbf{XO_1}$				
PS	XO_0	XO_1	$\mathbf{XO_2}$	$\mathbf{MU_0}$			
PFS	HC_1	HC_2	$\mathbf{HC_3}$	XO_0	XO_1	XO_2	$\mathbf{XO_3}$
TSP	$\mathbf{HC_2}$	XO_0	XO_1	XO_2	$\mathbf{XO_3}$		
VRP	HC_1	$\mathbf{HC_2}$					

3.2 ITC 2011: High School Timetabling Problem

3.2.1 Problem Description

The ITC 2011 problem instances [122] contain four components including *times*, *resources*, *events (meetings)* and *constraints* [121, 122, 167]. A time component represents an indivisible interval of time during which an event run. A resource represents the entity which attends an event. For example, teacher, room, student or class are resources. An event is a meeting between resources. A constraint is the condition that a solution should satisfy, if possible. In ITC 2011, 15 types of constraints are identified:

- **C01 Assign resource:** Assign resource to event.

- **C02 Assign time:** Assign time to event.
- **C03 Split events:** Forbid or allow splitting of event into sub-events under specific constraints.
- **C04 Distribute split events:** Split event into sub-events and distribute over a timetable by defining min or max duration time for each sub-event.
- **C05 Prefer resources:** Assign preferable resources to some events.
- **C06 Prefer times:** Assign preferable time slots to some events.
- **C07 Avoid split assignments:** Assign the same resource to set of events and specify whether a split assignment are desirable or not.
- **C08 Spread events:** Spread events evenly through the cycle in time between the given min and max in the solution.
- **C09 Link events:** Assign the same time to set of events.
- **C10 Avoid clashes:** Assign resource without having clashes.
- **C11 Avoid unavailable times:** Avoid assigning resources at unavailable times.
- **C12 Limit idle times:** Avoid having idle times for resources.
- **C13 Cluster busy times:** For number of days, resources must be busy.
- **C14 Limit busy times:** For number of times every day, resources must be busy.
- **C15 Limit workload:** Schedule the total workload without exceeding a limit.

In a standard fashion, constraints are separated into *hard* and *soft*. Each constraint has a boolean variable called *Required* to indicate whether the constraint is hard or soft. In the ITC2011 competition, such constraints are not strictly hard but are simply much more heavily penalised than the 'soft' constraints.

A candidate solution is evaluated in terms of two components: *feasibility* and *preferences*. The evaluation function computes the weighted hard and soft constraint violations for a given solution, where the weights are pre-defined in the input file representing a given instance, as *infeasibility* and *objective* values, respectively. The quality of a solution is denoted concatenating those two values as in $infeasibility{-}value.objective{-}value$ using sufficient number of digits in the objective-value part and filling with 0s if necessary. For

example, 10.000090 represents an infeasibility value of 10 and objective value of 90. For the comparison of algorithms, a solution is considered to be better than another one, if it has a smaller infeasibility value, or an equal infeasibility value and a smaller objective value. The minimum possible cost occurs whenever a *perfect solution* is obtained with a cost value of 0.000000, indicating that there are no constraint violations.

3.2.2 Test Instances

The participants of ITC 2011 tackled 35 instances of the high school timetabling problem, taken from schools in 10 countries. The high school timetabling instances were obtained across the world based on different education systems, where each problem came with its particular format. A unified format was required. Post et al. [167] proposed and used a common XML data format to represent a given problem instance of ITC 2011 as input. The ITC 2011 instances are therefore defined by a standard data format based on XML schema called XHSTT (XML High School Timetabling) [122, 167].

As a total of twenty one high school timetabling problem instances were made public during the first round of the competition. Eighteen hidden instances were used during the second round of the competition which are then made public and used for the third round of the competition. Table 3.4 summarises the main characteristics of all problem instances obtained across the world from different countries. These characteristics give some rough idea about the size of each instance, yet do not define a given problem fully as the importance of violating a given constraint is not provided. The ITC 2011 dataset can be downloaded from the competition website [121].

3.2.3 Low Level Heuristics

Ten low level domain-specific heuristics that are (mostly) fairly simple moves such as moving a task to a different resource, or swaps of events are designed and implemented to improve the initially generated solutions. The initial construction of a complete solution is performed using the general solver implemented by Jeff Kingston in the KHE library[1]. Note that the construction phase often gives a solution in which hard constraints are violated, and so the improvement phase also needs to improve the hard constraints.

The low level heuristics are divided into two sets; 8 mutational operators that do a randomised move by perturbing a given candidate solution in different ways, and 2

[1] `http://sydney.edu.au/engineering/it/~jeff/khe/`

TABLE 3.4: Characteristics of the problem instances used during three rounds of the competition

Round 1						
Instance - Country	Times	Teachers	Rooms	Classes	Students	Duration
BGHS98 - Australia	40	56	45	30		1564
SAHS96 - Australia	60	43	36	20		1876
TES99 - Australia	30	37	26	13		806
Instance1 - Brazil	25	8		3		75
Instance5 - Brazil	25	31		13		325
Instance7 - Brazil	25	33		20		500
StPaul - England	27	68	67	67		1227
ArtificialSchool - Finland	20	22	12	13		200
College - Finland	40	46	34	31		854
HighSchool - Finland	35	18	13	10		297
SecondarySchool - Finland	35	25	25	14		306
HighSchool1 - Greece	35	29		66		372
Patras 3rd HS 2010 - Greece	35	29		84		340
Preveza 3rd HS 2008 - Greece	35	29		68		340
Instance1 - Italy	36	13		3		133
GEPRO - Netherlands	44	132	80	44	846	2675
Kottenpark2005 - Netherlands	37	78	42	26	498	1272
Lewitt2009 - South Africa	148	19	2	16		838
Common to All Rounds						
Instance4 - Brazil	25	23		12		300
Instance6 - Brazil	25	30		14		350
Kottenpark2003 - Netherlands	38	75	41	18	453	1203
Rounds 2 and 3						
Instance2 - Brazil	25	14		6		150
Instance3 - Brazil	25	16		8		200
ElementarySchool - Finland	35	22	21	60		445
SecondarySchool2 - Finland	40	22	21	36		566
Aigio 1st HS 2010 - Greece	35	37		208		532
Instance4 - Italy	36	61		38		1101
Instance1 - Kosovo	62	101		63		1912
Kottenpark2005A - Netherlands	37	78	42	26	498	1272
Kottenpark2008 - Netherlands	40	81	11	34		1118
Kottenpark2009 - Netherlands	38	93	53	48		1301
Woodlands2009 - South Africa	42	40		30		1353
School - Spain	35	66	4	21		439
WesternGreeceUni3 - Greece	35	19		6		210
WesternGreeceUni4 - Greece	35	19		12		262
WesternGreeceUni5 - Greece	35	18		6		184

hill climbing operators that search their neighbourhoods for better solutions. The mutational heuristics return a solution after processing a given solution with no quality guarantee, while a hill climbing heuristic always returns a non-worsening solution, even if the returned solution is the same as the input.

Mutational move operators:

- **LLH0 ($\mathbf{MU}_0$):** swaps the start time of two randomly selected events. For example, assuming that the $Mathematics$ class meeting is assigned to the first time slot on Monday and the $History$ class meeting assigned to the third time slot on Friday, after the swap operation, $History$ is assigned to the first time slot on Monday, while $Mathematics$ to the third time slot on Friday.
- **LLH1 ($\mathbf{MU}_1$):** randomly selects an event and reschedules it to a random time. For example, assuming that the $Mathematics$ class meeting is assigned to the first time slot on Monday, after applying this heuristic, $Mathematics$ could be rescheduled to the last time slot on Friday.
- **LLH2 ($\mathbf{MU}_2$):** swaps the time of two randomly chosen events. If both events have the same duration, this heuristic operates like MH_1, but if their durations are not the same then the first chosen event is moved to the time slot right after the second event ends. For example, when swapping a $Mathematics$ class meeting with a duration of one assigned to the first time slot on Friday with a $History$ class meeting with a duration of two assigned to the second time slot on Friday, MH_3 moves the $Mathematics$ class to the third time slot on Friday, rather than the second time slot, and moves the $History$ class to the first time slot on Friday.
- **LLH3 ($\mathbf{MU}_3$):** selects a random resource element within an event and modifies its assignment randomly. For example, assuming that $Classroom1$ is assigned for the $Physics$ meeting, after applying this heuristic, $Classroom1$ can be reassigned for a meeting of $Mathematics$.
- **LLH4 ($\mathbf{MU}_4$):** swaps two random resources. For example, assuming that $Classroom1$ is assigned for $Mathematics$ and $Classroom2$ is assigned for $History$, after applying this heuristic, $Classroom1$ is assigned for the $History$ lesson while $Classroom2$ is assigned for the $Mathematics$ lesson.
- **LLH5 ($\mathbf{MU}_5$):** reassigns a randomly chosen resource element of an event to a random resource. For example, assuming that $Teacher1$ is assigned to teach $Mathematics$, after applying this heuristic, $Teacher1$ gets replaced by $Teacher8$.

- **LLH6 ($\mathbf{MU}_6$):** merges two class meetings of the same event and adjacent in time. For example, assuming that the *Biology* class meeting with a duration of two is assigned to the first time slot on Monday and another *Biology* class meeting with a duration of one is assigned to the third time slot on Monday, then after applying the heuristic, the *Biology* class meetings are merged into a single meeting with a duration of three starting at the first time slot on Monday.

- **LLH7 ($\mathbf{MU}_7$):** splits a randomly selected event requiring an assignment of a time block consisting of multiple time slots into two events with separate times with a fixed low probability of 0.1%. For example, assuming that a *Biology* class meeting is randomly chosen which has an assignment of a time block of two consecutive time slots, MH_7 divides the teaching of *Biology* into two separate (but still consecutive) time slots without changing their current assignments allowing future moves to operate on those two meetings separately.

Unlike most of the mutational operators, the 2 hill-climbing heuristics are capable of making quite large changes to a solution.

The hill climbing heuristics are themselves slightly non-standard. One of the operators [**LLH7 ($\mathbf{HC}_0$)**] is designed using neighbourhood structures based on ejection chains while the other operator [**LLH8 ($\mathbf{HC}_1$)**] is a type of first improvement hill climbing operator. Both hill climbing operators attempt to make moves which respect to a particular constraint type while hoping to improve upon the other types of constraint violations but might have a net worsening of the objective, however, then such worsening moves are rejected. For example, it may remove the violation of assigning a resource, but may introduce another violations to other constraints and increase the value of the evaluation function. If the cost of the new solution is improved, the repair terminates successfully. If not, the method calls itself recursively in an attempt to improve the quality of the solution; in this way a chain of coordinated changes is built up. If the recursive call fails to improve the quality, the method undoes the repair and returns the previous solution.

3.3 MISTA 2013 Challenge: Multi-mode Resource-constrained Multi-project Scheduling Problem

3.3.1 Problem Description

The problem consists of a set of *projects* $P = \{1, 2, \ldots, q\}$, where each project $p \in P$ is composed of a set of *activities*, denoted as A_p, a partition from all activities $A = \{1, 2, \ldots, n\}$. Each project $p \in P$ has a *release time* e_p, which is the earliest start time for the set of activities A_p.

The activities are interrelated by two different types of constraints: the *precedence constraints*, which force each activity $j \in A$ to be scheduled after all the predecessor activities in set $Pred(j)$ are completed; and the *resource constraints*, in which the processing of the activities is subject to the availability of resources with limited capacities. There are three different types of the resources: local renewable, local non-renewable and global renewable. Renewable resources have a fixed capacity per time unit. Non-renewable resources have a fixed capacity for the whole project duration. Global renewable resources are shared between all the projects while local resources are specified independently for each project.

$\mathscr{R}_p^\rho = \{1, 2, \ldots, |\mathscr{R}_p^\rho|\}$ is the set of local renewable resources associated with a project $p \in P$, and R_{pk}^ρ is the *capacity* of $k \in \mathscr{R}_p^\rho$, i.e., the amount of the resource k available at each time unit. $\mathscr{R}_p^\nu = \{1, 2, \ldots, |\mathscr{R}_p^\nu|\}$ is the set of local non-renewable resources associated with a project $p \in P$, and R_{pk}^ν is the capacity of $k \in \mathscr{R}_p^\nu$, i.e., the amount of the resource k available for the whole duration of the project. $\mathscr{G}^\rho = \{1, 2, \ldots, |\mathscr{G}^\rho|\}$ is the set of the global renewable resources, and G_k^ρ is the capacity of the resource $k \in \mathscr{G}^\rho$.

Each activity $j \in A_p$, $p \in P$, has a set of execution modes $\mathscr{M}_j = \{1, 2, \ldots, |\mathscr{M}_j|\}$. Each mode $m \in \mathscr{M}_j$ determines the duration of the activity d_{jm} and the activity resource consumption: r_{jkm}^ρ for each local renewable resource $k \in \mathscr{R}_p^\rho$, r_{jkm}^ν for each local non-renewable resource $k \in \mathscr{R}_p^\nu$ and g_{jkm}^ρ for each global renewable resource $k \in \mathscr{G}^\rho$.

Schedule $D = (T, M)$ is a pair of time and mode vectors, each of size n. For an activity $j \in A$, values T_j and M_j indicate the start time and the execution mode of j, respectively. Schedule $D = (T, M)$ is feasible if:

- For each $p \in P$ and each $j \in A_p$, the project release time is respected: $T_j \geq e_p$;

- For each project $p \in P$ and each local non-renewable resource $k \in \mathscr{R}_p^\nu$, the total resource consumption does not exceed its capacity R_{pk}^ν.
- For each project $p \in P$, each time unit t and each local renewable resource $k \in \mathscr{R}_p^\rho$, the total resource consumption at t does not exceed the resource capacity R_{pk}^ρ.
- For each time unit t and each global renewable resource $k \in \mathscr{G}_p^\rho$, the total resource consumption at t does not exceed the resource capacity G_k^ρ.
- For each $j \in A$, the precedence constraints hold: $T_j \geq \max_{j' \in Prec(j)} T_{j'} + d_{j'M_{j'}}$.

The objective of the problem is to find a feasible schedule $D = (T, M)$ such that it minimises the so-called *total project delay* (TPD), also referred to as $f_\mathrm{d}(D)$ in this study:

$$f_\mathrm{d}(D) = \left(\sum_{p \in P} \max_{j \in A_p} \left(T_j + d_{jM_j} \right) \right) - L \,, \tag{3.1}$$

where L is a lower bound (constant) calculated as

$$L = \sum_{p \in P} \left(\mathrm{CPD}_p + e_p \right) \,, \tag{3.2}$$

and CPD_p is a given pre-calculated value.

The tie-breaking secondary objective is to minimise the *total makespan* ($f_\mathrm{m}(D)$), which is the finishing time of the last activity:

$$f_\mathrm{m}(D) = \max_{j \in A} \left(T_j + d_{jM_j} \right) \,. \tag{3.3}$$

The objective functions $f_\mathrm{d}(D)$ and $f_\mathrm{m}(D)$ are combined into one function $f(D)$ that gives the necessary ranking to the solutions:

$$f(D) = f_\mathrm{d}(D) + \gamma f_\mathrm{m}(D) \,, \tag{3.4}$$

where $0 < \gamma \ll 1$ is a constant selected so that $\gamma f_\mathrm{m}(D) < 1$ for any solution D produced by the algorithm. In fact, we sometimes use $\gamma = 0$ to disable the second objective.

3.3.2 Test Instances

Three sets of instances have been used during the MISTA 2013 Challenge. These instances are produced by combining several multi-mode resource-constrained project scheduling problem (MRCPSP) instances. The MRCPSP instances are generated by the standard project generator ProGen [151]. Briefly, the construction of the instances requires a construction of network subject to a set of constraints, resource factor which reflects the average portion of resources per activity and resource strength to express the degree of availability of the resources. The format of the MRCMPSP data is based on the PSPLIB[2] MRCPSP data format. More about the generation of the instances can be found on [151] and the PSPLIB website.

Table 3.5 summarises the main characteristics of these instances. In the table, q is the number of projects, n is the number of activities, avg. d is the average duration of activities in all possible modes, avg. $|\mathscr{M}|$ is the average number of modes for each activity, avg. $|Pred|$ is the average number of the predecessor activities for each activity, avg. $|\mathscr{R}^{\rho}|$ is the average number of the local renewable resources per project, avg. $|\mathscr{R}^{\nu}|$ is the average number of the local non-renewable resources per project, $|\mathscr{G}^{\rho}|$ is the number of global renewable resources, avg. R^{ρ} is the average renewable resource capacities, avg. C^{ν} is the average non-renewable resource capacities, avg. G^{ρ} is the average global renewable resource capacities, avg. CPD is the average CPD per project and H is the upper bound on the time horizon. The information provided on the table gives some indication about the size of each instance.

3.3.3 Solution Construction

In designing an algorithm, there are two 'natural' representations to be used in the search for an assignment of activities to times:

Schedule-based: A direct representation using the assignment times of activities.

Sequence-based: This is based on selecting a total order of all the activities. Given such a sequence then a time schedule is constructed by taking the activities one at a time in the order of the sequence and placing each one at the earliest time that it will go in the schedule.

[2]PSPLIB benchmark: `http://129.187.106.231/psplib/`

TABLE 3.5: Instances characteristics in MISTA 2013

Instance	q	n	avg. d	avg. $\lvert\mathscr{M}\rvert$	avg. $\lvert Pred\rvert$	avg. $\lvert\mathscr{R}^{\rho}\rvert$	avg. $\lvert\mathscr{R}^{\nu}\rvert$	$\lvert\mathscr{G}^{\rho}\rvert$	avg. R^{ρ}	avg. C^{ν}	avg. G^{ρ}	avg. CPD	H
A-1	2	20	5.53	3	1.20	1	2	1	18.5	51.3	16.0	14.5	167
A-2	2	40	4.63	3	1.70	1	2	1	23.5	117.3	23.0	22.5	303
A-3	2	60	5.51	3	1.73	1	2	1	38.5	154.8	49.0	33.5	498
A-4	5	50	4.37	3	1.20	1	2	1	15.2	44.9	12.0	14.2	409
A-5	5	100	5.43	3	1.70	1	2	1	24.0	92.4	13.0	23.0	844
A-6	5	150	5.13	3	1.73	1	2	1	23.8	175.4	13.0	25.6	1166
A-7	10	100	6.03	3	1.20	0	2	2	0.0	48.4	11.5	16.8	787
A-8	10	200	5.67	3	1.70	0	2	2	0.0	110.8	22.5	24.6	1569
A-9	10	300	5.61	3	1.73	1	2	1	27.5	168.0	27.0	29.6	2353
A-10	10	300	5.53	3	1.73	1	2	1	25.9	158.2	15.0	30.7	2472
B-1	10	100	5.33	3	1.20	1	2	1	17.1	44.8	11.0	12.9	821
B-2	10	200	5.67	3	1.70	0	2	2	0.0	94.0	21.0	23.9	1628
B-3	10	300	5.52	3	1.73	1	2	1	28.5	144.4	28.0	29.5	2391
B-4	15	150	5.03	3	1.20	1	2	1	17.5	52.3	10.0	15.8	1216
B-5	15	300	6.02	3	1.70	1	2	1	20.7	99.6	17.0	22.5	2363
B-6	15	450	4.62	3	1.73	1	2	1	25.0	141.8	34.0	31.1	3582
B-7	20	200	4.87	3	1.20	1	2	1	14.7	49.6	10.0	15.4	1596
B-8	20	400	5.48	3	1.70	0	2	2	0.0	104.7	10.0	23.7	3163
B-9	20	600	5.31	3	1.73	1	2	1	26.6	154.8	10.0	30.1	4825
B-10	20	420	5.28	3	1.66	0	2	2	0.0	115.9	18.0	24.5	3340
X-1	10	100	5.53	3	1.20	0	2	2	0.0	47.6	12.5	14.9	783
X-2	10	200	5.53	3	1.70	1	2	1	24.0	105.6	14.0	23.0	1588
X-3	10	300	4.98	3	1.73	1	2	1	27.9	167.0	33.0	29.9	2404
X-4	15	150	5.70	3	1.20	0	2	2	0.0	54.5	13.5	14.9	1204
X-5	15	300	5.52	3	1.70	1	2	1	19.9	100.1	12.0	23.6	2360
X-6	15	450	5.49	3	1.73	1	2	1	24.6	163.7	20.0	29.9	3597
X-7	20	200	5.03	3	1.20	1	2	1	13.9	53.9	10.0	15.0	1542
X-8	20	400	5.53	3	1.70	1	2	1	22.2	104.5	15.0	24.5	3217
X-9	20	600	5.54	3	1.73	1	2	1	23.9	146.3	11.0	28.9	4699
X-10	20	410	5.30	3	1.65	1	2	1	20.0	101.2	10.0	24.1	3221

The schedule-based representation is perhaps the most natural for a mathematical programming approach, but we believe that it could make the search process difficult for a meta-heuristic method, in particular, generating a feasible solution at each step could become more challenging. Hence, we preferred the sequence-based representation, since it provides the ease of producing schedules that are both feasible and for which no activity can be moved to an earlier time without moving some other activities.

Sequence-based representation is a pair $S = (\pi, M)$, where π is a permutation of all the activities A, and M is a modes vector, same as in the direct representation. Permutation π has to obey all the precedence relations, i.e., $\pi(j) > \pi(j')$ for each $j \in A$ and $j' \in Pred(j)$. The modes vector is feasible if $M_j \in \mathscr{M}_j$ for each $j \in A$ and the local non-renewable resource constraints are satisfied for each project $p \in P$.

In order to evaluate a solution S, it has to be converted into the direct representation D. By definition, the sequence-based representation $S = (\pi, M)$ corresponds to a schedule produced by consecutive allocation of activities $\pi(1)$, $\pi(2)$, ..., $\pi(n)$ to the earliest available position. The corresponding procedure is formalised in Algorithms 8 and 9.

Algorithm 8: Schedule construction

```
1 Let S = (π, M) be the solution;
2 for i ← 1, 2, ..., n do
3 |   Let j ← π(i);
4 |   Schedule j in mode M_j to the earliest available position;
5 end
```

3.3.3.1 Construction Phase

This part describes how we construct an initial solution to be supplied to the later improvement phase. In the random constructor, the order (permutation) of the set of activities is randomly generated while respecting the precedence constraints. The set of modes associated with the activities are set in such a way that the local non-renewable resource constraints are not violated for any project. However, the constructor method that we have proposed is designed to not only produce feasible solutions, but also with the goal that they have a structure similar to that expected in good solutions. We observed that many such cases had an approximate ordering of the projects. That is, there would be time periods in the schedule when the general focus would be on one or few projects and during the schedule this focus would change between projects. Such a structure naturally arises because the primary objective is the delay-based TPD

Algorithm 9: Scheduling an activity to the earliest available position

Let j be the activity to be scheduled;
Let m be the mode associated with j; Let p be an index such that $j \in A_p$;
Calculate the earliest start time of j as $t_0 \leftarrow \max\left\{e_p, \max_{j' \in Prec(j)} (t_j + d_{pjm})\right\}$;
for $t \leftarrow t_0, t_0 + 1, t_0 + 2, \ldots$ **do**
    for $t' \leftarrow t, t + 1, \ldots, t + d_{pjm} - 1$ **do**
        for $k \in \mathscr{R}_p^\rho$ **do**
            Let a be the remaining capacity of k at t';
            if $r_{jkm}^\rho > a$ **then** proceed to the next t
        end
        for $k \in \mathscr{G}^\rho$ **do**
            Let a be the remaining capacity of k at t';
            if $r_{jkm}^\rho > a$ **then** proceed to the next t
        end
    end
end
Allocate activity j at t in mode m and update the remaining capacities;

rather than the overall makespan. Suppose that we look at the two projects that finish latest. It can well be that the penultimate project can move its last activities earlier by adjusting the activities of the last finishing project (except its very last activity) then this will improve the TPD. Unlike the makespan objective, the TPD objective encourages unfairness between the finish times of projects, and can drive some projects to finish as early as possible.

Overall, the structures were only partial orderings of the projects, but with more of a tendency for the latest projects to be critical. Consequently, our constructor attempts to create initial sequences that mimic such project (partial) orderings. Furthermore, we expect that only a partial ordering is needed because we can expect that the subsequent improvement phase can be expected to make small or medium size adjustments to the overall project ordering structure. However, the improvement phase would have more difficulty, and take more iterations, if the general structure of the project ordering were not good. Consequently, we decided that a reasonable approximation would be to use a 3-way partition of the projects taken to correspond to 'start', 'middle' and 'end' of the overall project time. We required the numbers of projects in each partition to be equal or with a difference of at most one when the total number of projects is not a multiple of 3.

The problem then is how to quickly select a good partition of the projects, and the

method we selected is a version of Monte-Carlo Tree Search (MCTS) methods [168]. The general idea of MCTS is to search a tree of possibilities, but the evaluation of leaves is not done using a predefined heuristic, but instead by sampling the space of associated solutions. The sampling is performed using multiple invocations of a "rollout" which is designed to be fast and unbiased. It needs to be fast so that multiple samples can be taken; also rather than trying to produce "best solutions" it is usually designed to be unbiased - the idea being that it should provide reliable branching decisions in the tree, but is not directly trying to find good solutions.

In our case, the tree search corresponds to decisions on which projects should be placed in which partition. The rollout is a fast way to sample the feasible activity sequences consistent with the candidate choice for the partition of the projects. Specifically, the tree search works in two levels; firstly to select the projects to be placed in the end partition and then to select the partition between the start and middle partitions.

The first stage considers 100 random choices for the partition of the projects, and then selects between these using 120 samples or the rollout. The rollout consists of two main stages:

1. Randomly select a total ordering of the activities consistent with the precedences and with the candidate partitioning. Specifically, within each partition we effectively consider a dispatch policy that randomly selects between activities that are available to be scheduled because their preceding activities (if any) are already scheduled.

2. Randomly select modes for the activities. If the result is not feasible then this can only be because of the mode selection causing a shortfall in some non-renewable resources. Hence, it is repaired using hill climbing on the space of mode selections. We use moves that randomly flip one mode at a time, and an objective function that measures the degree of infeasibility by the shortfall in resources. Since the non-renewable resources are not shared between projects, this search turned out to be fast and reliable.

The first stage ends by making a selection of the best partitioning, using the quality of the 25'th percentile of the final solution qualities (the best quartile) of the results of the rollouts. The 'end' partition of this best partition is then fixed to that of the best partition. The decision to fix the 'end' partition also arose out of the observation that in good solutions the end projects are least interleaved. The MCTS proceeds to the second

stage, and follows the same rollout procedure but this time to select the contents of the middle (and hence start) partitions.

3.3.4 Low Level Heuristics

In this section, all the low level heuristics are described in detail. These heuristics are restricted, as needed, to only generate feasible sequences. The low level heuristics (also referred to as neighbourhood operators or simply moves, depending on the algorithm which makes use of them) are categorised into three groups. This categorisation is mainly based on the common nature of the strategy the operators employ while manipulating the solution sequence.

The low level heuristics can be modified to a limited extent through the use of the "Intensity/Depth of heuristic" parameters. Changing the value of the parameter to a low level heuristic modifies the behaviour of the low level heuristic. The meaning of the parameter depends on the low level heuristic in question. For example, it could specify the extent to which a local search heuristic will modify the solution and the number of improving steps to be completed by the local search heuristics; or it could mean how many activities are changed by one call of the heuristic; or could mean the percentage of the activities that are destroyed and rebuilt.

The following provides the description of each category along with a detailed explanation of the underlying mechanism in each of the member operators.

3.3.4.1 Swap, Insert and Set Operators

Operators belonging to this category are simple swap, insertion and set operators with various coverage areas ranging from a single activity to multiple projects. These operators can be used within local search procedures and mutation operators. The following is the list of the operators utilised in our algorithm.

- SwapActivities: Swap two activities within the sequence. First a random activity j_1 is chosen. Positions of the last predecessor (pos_p) and the first successor (pos_s) of j_1 are then determined. The last predecessor of j_1 is the closest preceding activity with respect to activity j_1. Likewise, the first successor of j_1 is the closest succeeding activity of j_1. Subsequent to determining pos_p and pos_s, a second activity (j_2), is randomly chosen such that $pos_p < pos_{j_2} < pos_s$. That is to say

that the activity j_2 is positioned between the two premises pos_p and pos_s. The two activities j_1 and j_2 are then swapped.

- InsertActivity: Inserts a given activity into a new location in the solution sequence. Similar to what was described for SwapActivities, an activity is randomly selected and, if feasible, inserted into a location between the last predecessor and the first successor of the selected activity.

- SetMode: This move is also based on choosing an activity randomly, the mode of which is set randomly while guaranteeing that the new mode is different from the previous mode of the selected activity. This is of course the case when the activity has more than one mode. Otherwise, the move does not perform any operations, leaving the activity mode intact.

- FILS swapActivities/insertActivity/setMode: These are First Improvement Local Search (FILS) procedures based on the swapping, inserting or mode changing. Considering the FILS swapActivities, the selection process starts with choosing a random activity (j_1) based on a uniform distribution. The positions of the last preceding activity (pos_p) and first succeeding activity (pos_s) of j_1 are determined. A window (W) of length l is then considered and placed randomly on the solution sequence such that $pos_{W_{start}} > pos_p$ and $pos_{W_{end}} < pos_s$. W_{start} and W_{end} are the positions of the start and the end of the window respectively. Starting from the first activity that falls within the window boundaries and for each such activity (j_2) the operator swap is performed. In case the operation improves the objective value, the solution is accepted and returned. Otherwise, the operation is undone and we move on to the subsequent position in the window. The same procedure applies to FILS insertActivity and setMode. The length (l) of the window is a parameter and can be considered as the search depth.

- SwapTwoProjects: Swaps two randomly selected projects in the sequence.

- SwapNeighbourProjects: This move is similar to SwapTwoProjects with the difference that the two projects selected for swapping should be neighbouring in the solution sequence.

3.3.4.2 Project-wise Mutational Operators

The main idea behind the operators belonging to this category is to perturb the position of a number of activities within the sequence in a project-wise manner. The major

reason behind designing project-wise operators is that, according to our observations, good quality solutions roughly tend to cluster the activities of each project in a close vicinity of each other. Moreover, it is important to have the projects in the right order as such an order has a tremendous effect on the quality of the solution. Thus, the operators described below are designed to manipulate the sequence solution to achieve a solution in which activities are roughly clustered according to the project they belong to as well as manipulating the order of projects within the sequence.

- MutationOneExtreme: the activities of a randomly selected project are all collected and squeezed into a randomly-selected position in the sequence. This way, all the activities which belong to a certain project are placed in adjacency of each other.
- MutationOne: Shifts all the activities of a randomly selected project by a number of positions in the sequence. The scale of the shift (the number of positions by which the activities are shifted) is chosen to be a random number which varies between the position of the first and last activities of the selected project.
- MoveProjects: Extracts the sequence of the projects in the solution (based on the positions of the last activities in each project), selects several consequent projects, and then moves them to either the beginning or end of the sequence.

3.3.4.3 Ruin & Recreate Operators

The Ruin&Recreate (R&R) operators consist of moves in which a list of activities is selected based on a specific distribution, forming the R&R list. The distribution with which the activities in the R&R list are chosen varies according to the type of the move (this is explained later in this section). The selected R&R activities are all guaranteed to be different. The position of the selected activities on the solution sequence are considered to be vacant, ready to be occupied as the move completes its operation. Subsequent to activity selection, each operator performs a move which can be restricted to moving the selected activities or changing their respective modes or both. That is, three options are incorporated into the move:

- Moving activities: prior to moving the selected activities, they are reshuffled randomly. Also, a matrix of precedence feasibilities of the activities within the list is

utilised which is constructed during an earlier pre-processing phase. The precedence relationship of the selected activities in the R&R list constitutes a directed graph where there is an edge from a node to its successor(s). There are always nodes without incoming edges (nodes for which no predecessor can be found among those activities in the list). Thus, when moving activities, the R&R list is scanned for the first such activity with zero predecessors. This activity is then placed in the sequence, if feasible, and the graph is updated accordingly where new nodes without any preceding activities emerge. This procedure continues recursively until no activities remain.

- Changing modes: for each activity in the R&R list, a new random mode is chosen. In case the list contains one or more activities with infeasible modes (resulting in negative availability of the local non-renewable resource), the entire sequence of chosen modes is rejected and another sequence of modes is chosen randomly. This process continues until a feasible set of modes is found for the activities within the list.

- Both: changing modes followed by moving activities, both as described above.

A number of variants of R&R operators is designed where the major distinguishing feature among the variants is the distribution with which the activities of the R&R list are chosen. Please note that the higher level heuristic which makes its choice among the move operators of this category, considers each option of an R&R operator as a separate move. In what follows, the activity selection strategy in each move has been described. Furthermore, the intuition behind employing such strategies is explained.

- MoveUniform: As the name suggests the R&R activities are selected according to a uniform distribution, giving each activity an equal chance to move within the sequence and/or change mode.

- MoveLocal: The activities are selected using a specific and non-uniform distribution centred around a controllable position (p), and with a controllable width (w), within the sequence. Specifically, a random activity (j) is selected according to the following probability.

$$p = \frac{1}{\frac{t_j/T - pos}{width} + 1} \tag{3.5}$$

where, t_j is the scheduled activity start time and T is the overall time-span.

- MovePROJECT: The activity selection of this move is biased towards a randomly selected project.

- MoveBiasedGlobalResource: While selecting the activities, this move favours those activities which are scheduled at a time in which remaining capacity on global resources is higher. Hence, This move aims at avoiding the under-usage of global resources.

  We start by selecting a random activity j by employing a roulette wheel selection strategy. In other words, the probability of selecting an activity is proportional to the global resource consumption ratio.

$$\frac{\sum_{k=1}^{|\mathscr{G}^\rho|} g_{jkm}^\rho}{\sum_{k=1}^{|\mathscr{G}^\rho|} G_\rho^k} \tag{3.6}$$

  where $\sum_{k=1}^{|\mathscr{G}^\rho|} g_{jkm}^\rho$ is the available global resources for activity j in mode m. $\sum_{k=1}^{|\mathscr{G}^\rho|} G_\rho^k$ is the sum of global resource capacities.

- MoveEndBiased: Favouring the activities with a position close to the end of projects, this move aims at polishing the project endings. Consider a project which is neatly scheduled within the sequence such that the majority of its activities are adjacent to each other. Often, such a schedule has a lower TDP value compared to a project whose activities are scattered all over the time horizon. Hopefully, biasing the activity selection process towards the project endings leads to a sequence in which a larger number of activities belonging to the same project are adjacently positioned. MoveEndBiased, employs such an strategy. Selecting activities is based on the roulette wheel approach. The probability of selecting an activity $j \in A_p$ is proportional to the ratio $pos_p^j/|A_p|$. pos_p^j is the position of activity j within the project p while $|A_p|$ is the number of activities in project p. Obviously, the chances of selecting an activity are higher if it is positioned closer to the project ending.

3.3.5 Improvement Phase

Most of the time (more than 95% of the given 5 minutes) our algorithm spends on improving the initial solutions. We use a multi-thread implementation of a simple memetic algorithm with a local search procedure based on a multi-stage hyper-heuristic which controls low level moves. The multi-stage hyper-heuristics are explained in Chapter 4.

The improvement phase of our algorithm is controlled by a simple multi-threaded MA which effectively manages the solution pool and utilises all the cores of the CPU. Our MA [169] is based on quantitative adaptation at a local level. Within the MA, we use a powerful local search procedure that takes significant time to converge. To achieve a sufficient number of generations, we keep the population size small. In particular, we maintain one solution per CPU core, i.e., 8 solutions, since the test machine has an Intel i7 CPU with 8 virtual cores. Each local search run takes exactly 5 seconds and is performed on a dedicated core. Since local search takes virtually all computational time, this simple parallelisation provides over 95% CPU utilisation.

Because of the small population size and limited number of generations, we decided to use a simple version of the MA, see Algorithm 10. The population consists of solutions S_i, $i = 1, 2, \ldots, 8$. The memetic algorithm uses the following subroutines:

- *Construct*() returns a new random solution generated according to the initial partial project sequence as described in Section 3.3.3.1.
- *Accept*(S_i) returns *true* if the solution S_i is considered 'promising' and *false* otherwise. The function returns *false* in two cases: (1) if $f(S_i) > 1.05 f(S_{i'})$ for some $i' \in \{1, 2, \ldots, 8\}$ or (2) if the solution was created at least three generations ago and S_i is among the worst three solutions. Ranking of solutions is performed according to $f_{\mathrm{d}}(S_i) + idle$, where *idle* is the number of consecutive generations that did not improve the solution S_i.
- *Select*(S) returns a solution from the population chosen with the simple tournament selection with two individuals.
- *Mutate*(X) returns a new solution produced from X by applying a mutation operator. The mutation operator to be applied is selected randomly and uniformly among the available options.

The algorithm contains five mutation operators:

- Apply R&R for both positions and modes using the MoveLocal selection mode. Repeat the procedure 20 times, each time selecting 3 activities near a randomly and uniformly chosen position $p \in [0, 1]$. The selection 'width' w is 0.1.
- SwapNeighbourProjects.

- MovePROJECTS for one project being moved to the end of the sequence.
- MovePROJECTS for two projects being moved to the beginning of the sequence.
- MovePROJECTS for three projects being moved to the beginning of the sequence.

Recall that the objective consists of two competing components. Indeed, minimisation of the total makespan favours solutions with projects running in parallel as such solutions are more likely to achieve higher utilisation of the global resources. At the same time, minimisation of the TPD favours solutions with the activities grouped by projects. Hence, the second objective creates a pressure for the local search that pushes the solutions away from the local minima with regards to the first objective. To avoid this effect, we disable the second objective ($\gamma \leftarrow 0$) and enable it only after 70% of the given time is elapsed.

Algorithm 10: Improvement Phase

```
1  γ ← 0;
2  for i ← 1, 2, ..., 8 do
3  |  S_i ← Construct();
4  end
5  while there is time remaining do
6  |  if elapsedtime ≥ 0.7giventime then
7  |  |  γ ← 0.000001;
8  |  end
9  |  for i ← 1, 2, ..., 8 (multi-threaded) do
10 |  |  S_i ← LocalSearch(S_i);
11 |  end
12 |  for i ← 1, 2, ..., 8 do
13 |  |  if Accept(S_i) = false then
14 |  |  |  X ← Select(S);
15 |  |  |  S_i ← Mutate(X);
16 |  |  end
17 |  end
18 end
```

Seventeen low level heuristics are controlled by the multi-stage hyper-heuristics. They are partitioned into three sets as LLH_{small}, LLH_{medium}, LLH_{large} considering the number of activities processed (e.g., number of swaps) by a given heuristic. Medium moves are the local search heuristics and ruin & recreate operators.

- LLH_{small}: {**LLH0 (MU_0):** SwapActivities, **LLH1 (MU_1):** InsertActivity, **LLH2 (MU_2):** SetMode}.

- $\mathbf{L}\text{LH}_{medium}$: {**LLH3 ($\mathbf{RC}_0$):** MoveUniform(changing modes), **LLH4 ($\mathbf{RC}_1$):** MoveUniform(both), **LLH5 ($\mathbf{RC}_2$):** MoveLocal(changing modes), **LLH6 ($\mathbf{RC}_3$):** MoveLocal(both), **LLH7 ($\mathbf{RC}_4$):** MoveBiasedGlobalResource(both), **LLH8 ($\mathbf{RC}_5$):** MoveEndBiased(both), **LLH9 ($\mathbf{RC}_6$):** MoveProject(both), **LLH10 ($\mathbf{HC}_0$):** FILS SwapActivities, **LLH11 ($\mathbf{HC}_1$):** FILS InsertActivity, **LLH12 ($\mathbf{HC}_2$):** FILS setMode}.
- $\mathbf{L}\text{LH}_{large}$: { **LLH13 ($\mathbf{MU}_3$):** MoveProjects, **LLH14 ($\mathbf{MU}_4$):** SwapTwoProjects, **LLH15 ($\mathbf{MU}_5$):** MutationOneExtreme, **LLH16 ($\mathbf{MU}_6$):** MutationOne}.

3.4 SolveIT International Optimisation Competition: Constructing Magic Squares

3.4.1 Problem Description

A magic square of order n is a square matrix of size nxn, containing each of the numbers 1 to n^2 exactly once, in which the n numbers in all columns, all rows, and both diagonals add up to the magic number $M(n)$. This constant is given by Equation 3.7:

$$M(n) = n(n^2 + 1)/2 \tag{3.7}$$

As an example, the magic square of order 3 is shown below:

$$\begin{bmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{bmatrix}$$

A formal formulation of the magic square problem is as follows. Given a magic square matrix A of order n such that

$$A_{n\times n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$$

where $a_{i,j} \in \{1, 2, ..., n^2\}$ for $1 \leq i, j \leq n$ and $a_{i,j} \neq a_{p,q}$ for all $i \neq p$ and $j \neq q$

subject to

$$\sum_{i=1}^{n} a_{i,j} = M(n),\ \sum_{j=1}^{n} a_{i,j} = M(n),\ \sum_{i=1}^{n} a_{i,(n+1-i)} = M(n) \text{ and } \sum_{i=1}^{n} a_{i,i} = M(n)$$

A constraint version of the magic squares problem requires for a given instance of size $n \geq 10$ that the solution matrix must have a contiguous sub-matrix $S_{3\times3}$ to be placed at a given location (i, j) in $A_{n\times n}$:

$$S_{3\times3} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The objective (cost) function measures the sum of absolute values of the distance from the Magic number for each column, row and diagonal. Hence, the problem can be formulated as a minimisation problem in which the goal is to *minimise* the objective function value in Equation 3.8. The magic square is found if the objective function value is 0.

$$\begin{aligned} g(A_{n\times n}) = &\sum_{i=1}^{n} \left| \sum_{j=1}^{n} a_{i,j} - M(n) \right| + \sum_{j=1}^{n} \left| \sum_{i=1}^{n} a_{i,j} - M(n) \right| \\ &+ \left| \sum_{i=1}^{n} a_{i,(n+1-i)} - M(n) \right| + \left| \sum_{i=1}^{n} a_{i,i} - M(n) \right| \end{aligned} \tag{3.8}$$

Equation 3.9 describes the objective function value after imposing the contiguous sub-matrix $S_{3\times3}$.

$$f(A_{n\times n}, i, j) = \begin{cases} g(A_{n\times n}) & \text{if } S_{3\times3} \text{ placed at the position } (i, j) \text{ in } A; \\ \infty & \text{otherwise.} \end{cases} \tag{3.9}$$

It is possible to obtain different magic squares of a given order. The following squares are examples of the two constrained version of the magic square of order 10:

$$\begin{bmatrix} 82 & 46 & 71 & 1 & 2 & 3 & 44 & 72 & 93 & 91 \\ 69 & 63 & 98 & 4 & 5 & 6 & 94 & 62 & 18 & 86 \\ 95 & 77 & 33 & 7 & 8 & 9 & 52 & 92 & 74 & 58 \\ 96 & 45 & 41 & 90 & 31 & 57 & 47 & 17 & 39 & 42 \\ 56 & 88 & 78 & 36 & 70 & 48 & 79 & 13 & 21 & 16 \\ 34 & 30 & 24 & 100 & 65 & 76 & 64 & 22 & 55 & 35 \\ 27 & 61 & 14 & 43 & 68 & 81 & 29 & 97 & 59 & 26 \\ 12 & 20 & 32 & 73 & 84 & 99 & 37 & 23 & 38 & 87 \\ 15 & 50 & 60 & 85 & 89 & 75 & 10 & 40 & 28 & 53 \\ 19 & 25 & 54 & 66 & 83 & 51 & 49 & 67 & 80 & 11 \end{bmatrix} \begin{bmatrix} 80 & 35 & 97 & 1 & 2 & 3 & 98 & 70 & 99 & 20 \\ 73 & 62 & 53 & 4 & 5 & 6 & 74 & 56 & 88 & 84 \\ 83 & 38 & 23 & 7 & 8 & 9 & 96 & 77 & 72 & 92 \\ 52 & 61 & 31 & 95 & 54 & 82 & 29 & 13 & 24 & 64 \\ 45 & 65 & 91 & 75 & 93 & 66 & 12 & 22 & 17 & 19 \\ 28 & 37 & 39 & 57 & 89 & 30 & 14 & 76 & 87 & 48 \\ 41 & 63 & 33 & 21 & 90 & 78 & 11 & 50 & 47 & 71 \\ 27 & 86 & 55 & 100 & 15 & 79 & 69 & 46 & 10 & 18 \\ 42 & 26 & 67 & 60 & 68 & 58 & 59 & 51 & 25 & 49 \\ 34 & 32 & 16 & 85 & 81 & 94 & 43 & 44 & 36 & 40 \end{bmatrix}$$

The International Optimisation Competition hosted by SolveIT for constructing magic square used the following performance measure to determine the best approach. The largest magic square that an algorithm constructs in one minute was considered to be the best algorithm.

All computational experiments are performed on small instances from n=10 up to 23 with increments of 1 and large instances from n=25, 50, 75, 100 up to 2600 with increments of 100, unless mentioned otherwise. 2600 is chosen as the maximum order for the magic squares problem, as the winning approach of the magic square competition was able to solve a magic squares problem of order 2600 as the largest instance under a minute on the competition computer.

3.4.2 Low Level Heuristics

A candidate solution is encoded using a direct representation in the form of a matrix. The objective (cost) function is described in Equation 3.9.

3.4.2.1 The Late Acceptance Hill Climbing Approach

The winner approach of the magic squares competition is a late acceptance hill climbing approach, denoted as LAHC, employed two different methods each with a different set of heuristics based on the size of a given problem. The first set is used on small problems, where a magic square of odd order less than or equal 23, and a magic square of even

order less than or equal 18. The second set is used on large problems where a magic square of order 20, 22 or larger than 23.

Small Problems

L is set to 1000. Initially, the square is filled randomly and the constraint sub-matrix $S_{3\times3}$ is fixed at its right location (i, j). Only one heuristic is applied and it is designed so as not to violate the proposed constraint[3], which swaps two randomly selected entries.

Large Problems

The approach uses a nested mechanism to construct the magic square. The square is divided into several sub-matrices called *Magic Frames* with size of $l \times l$ and $l \leq n$, where only the border two rows and two columns are non-zero. The sum of numbers at the border rows and columns are equal to the magic number $M(l)$. The sum of numbers in other rows, columns and diagonals are equal to $l \times l + 1$. The magic square constructed by recursively inserting the magic frames or by placing a smaller magic square inside the magic frame. Example of magic frame of size $l = 4$:

$$\begin{bmatrix} 7 & 2 & 14 & 11 \\ 16 & 0 & 0 & 1 \\ 5 & 0 & 0 & 12 \\ 6 & 15 & 3 & 10 \end{bmatrix}$$

Initially, the magic frame is filled randomly with the necessary set of numbers and their counterparts (e.g. 16 and its counterpart 1 as shown in the above example). The constraint sub-matrix $S_{3\times3}$ is fixed at its right location (i, j) if the frame contains some of them. The L is set to 50000. The evaluation function of constructing the magic frames measures the sum of absolute values of the distance from the Magic number from the sum of the first row and the sum of the first column numbers. The heuristics are designed so as not to violate the constraint. The heuristics are described as follows:

- **H**0: swap randomly with its counterpart (e.g. swap 16 and 1 shown in the above magic frame).
- **H**1: swap randomly two entries and their counterparts (e.g. swap 3 with 5 and 12 with 14 shown in the above magic frame).

[3] `http://www.cs.nott.ac.uk/~yxb/IOC/LAHC_MSQ.pdf`

The LAHC approach selects one of the two heuristics randomly with H2 has a higher probability to be selected.

If the contiguous submatrix $S_{3\times3}$ is close to the border, then we only need to construct magic frames starting from the outer border until we cover the contiguous submatrix, then apply the well known magic square construction methods to fill the unfilled matrix. If the contiguous submatrix is placed deeply inside, then the following swap move is applicable. Considering the four vertices of the matrix P1, P2, P3 and P4, if P1+P2=P3+P4 and they are not in any of the both diagonals, then it is possible to swap P1 with P3 and P2 with P4 without violating the magic constraints. By using this property, the contiguous submatrix $S_{3\times3}$ can be placed close to the border and then moved into the location (i, j)[4].

3.4.2.2 Set of Low Level Heuristics

Similar to the LAHC approach, two different sets of low level heuristics based on the size of the problem are employed. The first set is applicable to the small size of the problem (magic square of odd order less than or equal 23, and a magic square of even order less than or equal 18); and the second set to large size of the problem.

First Set of Low Level Heuristics

Initially, the square is filled randomly and the constraint sub-matrix $S_{3\times3}$ is fixed at its right location at (i, j). Nine low level heuristics are implemented. The low level heuristics randomly modify a complete solution in different ways while respecting the given constraint.

- **LLH0 ($\mathbf{MU}_0$):** swap two entries to fix the magic number violation by trying to select an entry that is not in a row, column or diagonal satisfying the magic rule. Then swap this entry with another entry so as to satisfy, hopefully, the magic rule for the selected row, column or diagonal.
- **LLH1 ($\mathbf{MU}_1$):** select two rows, columns or diagonals randomly to swap as a whole.
- **LLH2 ($\mathbf{MU}_2$):** select largest sum of row, column or diagonal and smallest sum of row, column or diagonal and swap the largest element from the first with the smallest in the second.

[4] `http://www.cs.nott.ac.uk/~yxb/IOC/LAHC_MSQ.pdf`

- **LLH3 ($\mathbf{MU}_3$):** similar to LLH0. The only difference is that the process is repeated until satisfying the magic rule for the selected row, column or diagonal; or until no improvement is observed.

- **LLH4 ($\mathbf{MU}_4$):** select two rows randomly k and l, fix violations by swapping entries on a single column s for the rows [32]. The swap occurs if and only if:

$$\sum_{j=1}^{n} a_{k,j} - M(n) = M(n) - \sum_{j=1}^{n} a_{l,j} = a_{k,s} - a_{l,s},\ k \neq l$$

  Similarly, for two randomly selected columns k and l, the swap will occur if:

$$\sum_{i=1}^{n} a_{i,k} - M(n) = M(n) - \sum_{i=1}^{n} a_{i,l} = a_{s,k} - a_{s,l},\ k \neq l$$

- **LLH5 ($\mathbf{MU}_5$):** swap two randomly selected entries which are not on the row, column or diagonal that satisfy the magic number rule.

- **LLH6 ($\mathbf{MU}_6$):** select two rows randomly k and l, fix the violations by swapping entries on two columns s and t separately for the rows [32], where $k \neq l$, $s \neq t$ and a swap occurs if and only if:

$$\sum_{j=1}^{n} a_{k,j} - M(n) = M(n) - \sum_{j=1}^{n} a_{l,j} = a_{k,s} - a_{l,s} + a_{k,t} - a_{l,t}$$

  Similarly, for two randomly selected columns k and l, the swaps will occur if:

$$\sum_{i=1}^{n} a_{i,k} - M(n) = M(n) - \sum_{i=1}^{n} a_{i,l} = a_{s,k} - a_{s,l} + a_{t,k} - a_{t,l}$$

- **LLH7 ($\mathbf{MU}_7$):** fix violations on a diagonal as much as possible. Mathematically [32]: for $i, j = 1, 2, ..., N$ and $i \neq j$:

  Swap $a_{i,i}$ with $a_{j,i}$ and $a_{i,j}$ with $a_{j,j}$ if:

$$a_{i,i} + a_{i,j} = a_{j,i} + a_{j,j} \text{ and } (a_{i,i} + a_{j,j}) - (a_{i,j} + a_{j,i}) = \sum_{i=1}^{n} a_{i,i} - M(n)$$

Swap $a_{i,j}$ with $a_{(n+1-j),j}$ and $a_{i,(n+1-i)}$ with $a_{(n+1-j),(n+1-i)}$ if:

$$a_{i,j} + a_{i,(n+1-i)} = a_{(n+1-j),j} + a_{(n+1-j),(n+1-i)} \text{ and}$$
$$(a_{i,(n+1-i)} + a_{(n+1-j),j}) - (a_{i,j} + a_{(n+1-j),(n+1-i)}) = \sum_{i=1}^{n} a_{(n+1-i),i} - M(n)$$

Swap row i and j if:

$$(a_{i,i} + a_{j,j}) - (a_{i,j} + a_{j,i}) = \sum_{i=1}^{n} a_{i,i} - M(n) \text{ and}$$
$$(a_{i,(n+1-i)} + a_{j,(n+1-j)}) - (a_{i,(n+1-j)} + a_{j,(n+1-i)}) = \sum_{i=1}^{n} a_{(n+1-i),i} - M(n)$$

Swap column i and j if:

$$(a_{i,i} + a_{j,j}) - (a_{i,j} + a_{j,i}) = \sum_{i=1}^{n} a_{i,i} - M(n) \text{ and}$$
$$(a_{(n+1-i),i} + a_{(n+1-j),j}) - (a_{(n+1-j),i} + a_{(n+1-i),j}) = \sum_{i=1}^{n} a_{(n+1-i),i} - M(n)$$

Swap row i and $(n+1-i)$ if:

$$(a_{i,i} + a_{(n+1-i),(n+1-i)}) - (a_{i,(n+1-i)} + a_{(n+1-i),i}) = \sum_{i=1}^{n} a_{i,i} - M(n)$$
$$= M(n) - \sum_{i=1}^{n} a_{(n+1-i),i}$$

- **LLH8 ($\mathbf{MU}_8$):** select the row, column or diagonal with the largest sum and row, column or diagonal with the lowest sum and swap each entry with a probability of 0.5.

Second Set of Low Level Heuristics

The second set of low level heuristics has only two low level heuristics and are applicably to relatively large size of the problems. The same construction and evaluation methods developed by LAHC are used. The approach uses a nested mechanism to construct the magic square by dividing the matrix into magic frames just as explained previously. The same heuristics which are used by LAHC to construct the magic frames are used as a

low level heuristics for the hyper-heuristic framework, **LLH0 ($\mathbf{MU}_0$)** is H0 and **LLH1 ($\mathbf{MU}_1$)** is H1.

3.5 Summary

This chapter describes the problems dealt with in the study. The cross domain heuristic search over six HyFlex problem domains, high school timetabling, multi-mode resource-constrained multi-project scheduling, and construction of magic squares were subjects of recent competitions of CHeSC 2011, ITC 2011, MISTA 2013 and SolveIT 2011, respectively. Hence, each domain has its own problem instances, each with a different characteristic. A selection hyper-heuristic performing a single point based search requires design and implementation of high level components, such as heuristic selection and move acceptance that will operate on the domain. However, it is assumed that problem domain components, such as, low level heuristics and initialisation method, do exist for the given domain. CHeSC 2011 comes with the implementation of six problem domains and relevant instances, hence users can solely focus on the hyper-heuristic development. As for rest of the problem domains, relevant domain level components are developed for use by the selection hyper-heuristics. Hence, the formulation of the problem, design and implementation details, such as, initialisation (initial solution construction) method and low level heuristics for each domain are presented in this chapter. Multi-stage hyper-heuristics that utilise the domain level components are described in the following chapter.

Chapter 4

Multi-stage Hyper-heuristics

Under a single point based search framework, a selection hyper-heuristic chooses a heuristic from a predefined set of low level heuristics and applies it on a candidate solution. The new solution is then considered and a decision is made whether it will be accepted or not. If accepted, the new solution replaces the current solution and the search continues iteratively [2]. A hyper-heuristic aims to exploit the strengths (and avoid the weaknesses) of different heuristics which perform differently on different problem instances by mixing/controlling those heuristics and utilising each heuristic at different steps of the search process. It has been observed that a selection hyper-heuristic may perform differently at different stages of the search process and there is a strong empirical evidence indicating that the choice of heuristic selection and move acceptance combination influences the overall performance of a hyper-heuristic [8, 23]. Hence, this study attempts to address whether it is viable to mix/control multiple hyper-heuristics for an improved performance of the overall search methodology. A general multi-stage hyper-heuristic framework which allows the use of multiple hyper-heuristics at different stages of the search process is described in this chapter. Additionally, a set of multi-stage hyper-heuristics designed based on the proposed framework is presented as provided in Table 4.1. Some of the proposed approaches operate based on the observation that not all low level heuristics for a problem domain would be useful at any point of the search process.

TABLE 4.1: The multi-stage hyper-heuristic approaches proposed in this thesis

Method	Label	Problem Domain	Reference
Greedy-gradient - Simulated Annealing Hyper-heuristic	GGHH	University course timetabling problem HyFlex problems	Kalender et al. [28, 29]
Dominance-based Random Descent/Gradient Hyper-heuristic with Naïve Move Acceptance	DRD	HyFlex problems	Özcan and Kheiri [170]
Robinhood Hyper-heuristic with an Adaptive Threshold Acceptance	RHH	HyFlex problems	Kheiri and Özcan [171]
Selection Hyper-heuristic with an Adaptive Threshold Acceptance	HySST	High school timetabling problem HyFlex problems	Kheiri et al. [43, 172]
Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance	DRW	Project scheduling problem HyFlex problems	Asta et al. [155, 173]
Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation	MSHH	HyFlex problems Constructing magic square problem	Kheiri and Özcan [174]

4.1 A Multi-stage Hyper-heuristic Framework

The traditional *single-stage* selection hyper-heuristic framework employs a single heuristic selection and a single move acceptance method. If a different hyper-heuristic component is used at any stage during the search process, this constitutes a different *stage* enabling the design of multi-stage hyper-heuristics as illustrated in Figure 4.1. Allowing the use of multiple hyper-heuristic components interchangeably under a multi-stage framework opens up richer design options, such as the possibility of having several hyper-heuristics controlling different sets of low level heuristics cooperatively. A multi-stage framework requires inclusion of an additional upper level which will be referred to as *multi-stage level* within the selection hyper-heuristic framework as shown in Figure 4.1. The multi-stage level allows the transition between available hyper-heuristics and their automated control at different points during the search process. We took into account the criteria of designing a hyper-heuristic defined in [12] such that, a hyper-heuristic should "(1) be fast to implement, (2) require far less expertise in either the problem domain or heuristic methods, and (3) robust enough to effectively handle a range of problems". The designed multi-stage hyper-heuristic framework satisfies all these previously suggested design criteria. The proposed multi-stage hyper-heuristic methodologies which are described in the following sections mostly combine different ideas from the literature. Even though the multi-stage hyper-heuristics designed based on the framework are relatively simple, the results indicate their effectiveness as solution methods. Algorithm 11 provides the pseudocode of a multi-stage hyper-heuristic algorithm based on the framework in Figure 4.1.

In the following sections, we describe six novel multi-stage hyper-heuristics (Table 4.1) based on the proposed framework. Each multi-stage hyper-heuristic exhibits a variety of characteristics reflecting the capability and flexibility of the framework:

(1) The multi-stage level in GGHH, DRD, HySST, DRW and MSHH controls two different hyper-heuristics, while it controls three hyper-heuristics in RHH.

(2) GGHH, DRD, DRW and MSHH control two hyper-heuristics in such a way that one of them reduces the set of low level heuristics adaptively keeping the ones that are expected to perform relatively well, while the other hyper-heuristic uses this information and applies only the reduced set of low level heuristics to improve the candidate solution at a stage. In RHH and HySST, all the hyper-heuristics controlled at the multi-stage level use a fixed number of low level heuristics at each stage during the search process.

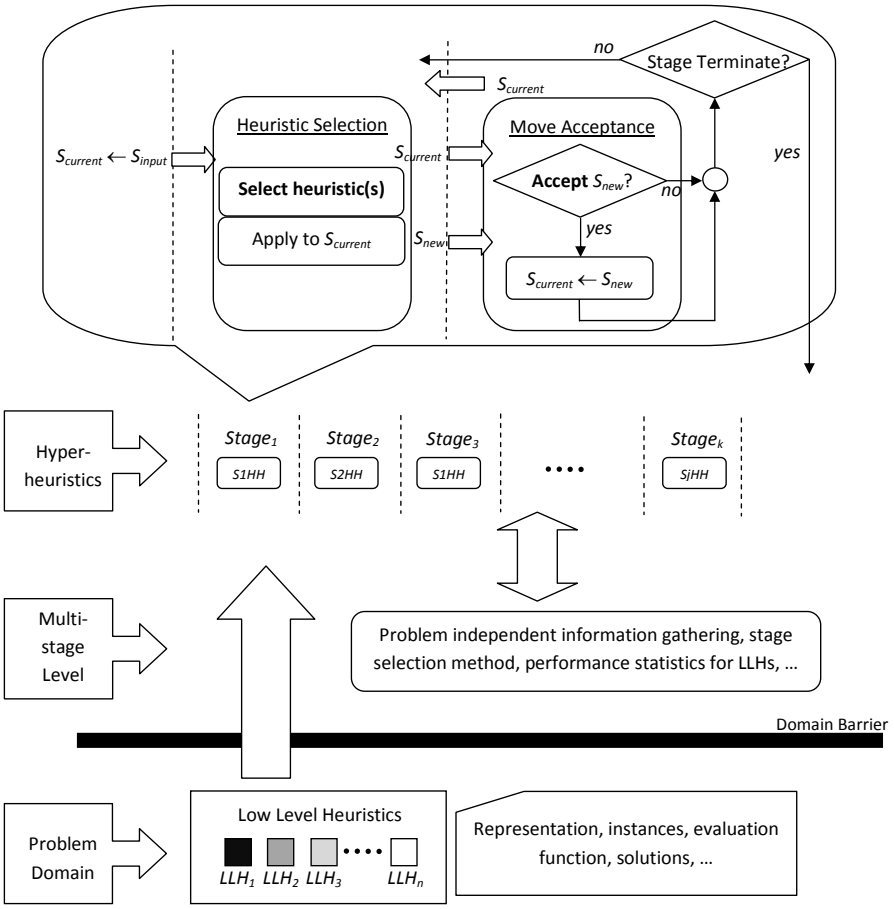


FIGURE 4.1: A multi-stage hyper-heuristic framework

(3) GGHH, DRD, DRW and MSHH methods ignore the nature of the low level heuristics, while HySST and RHH require that information and make use of the type of the heuristic whether it is mutational or hill climber while performing search.

(4) Each hyper-heuristic in RHH is applied for a fixed duration of time. The transition between stages for RHH is *static*. On the other hand, the transition between stages in which a different hyper-heuristic is used occurs *adaptively* in GGHH, DRD, HySST, DRW and MSHH.

(5) The transition between stages in DRD and MSHH is *stochastic*, i.e., a set of conditions has to be satisfied along with a certain probability for the following stage with a different hyper-heuristic to start. In GGHH, HySST and DRW, the transition is *non-stochastic*, i.e., it is sufficient for a stage with a different hyper-heuristic to start its execution given a set of conditions are satisfied.

Algorithm 11: Pseudocode of the multi-stage hyper-heuristic framework

```
1  Let HH = {S1HH, S2HH, ..., SjHH} represent set of all hyper-heuristics;
2  Let S_input represent set of input solutions;
3  Let S_output represent set of output solutions;
4  Let s_best represent the best solution;
5  repeat
6      SiHH ← SelectHyperHeuristic(HH);
7      Update1();      /* set/update relevant parameter/variable values before
       entering into a stage */
8      while notSatisfied(SiHHTerminationCriteria) do
9          S_output, s_best ← ApplyHyperHeuristic(SiHH, S_input);
10         Update2(); /* set/update relevant parameter/variable values during a
           stage */
11     end
12     Update3();      /* set/update relevant parameter/variable values after
       finishing a stage */
13 until TerminationCriterionSatisfied();
14 return s_best;
```

In the following sections, we discuss each multi-stage hyper-heuristic in Table 4.1.

4.2 Greedy-gradient - Simulated Annealing Hyper-heuristic (GGHH)

4.2.1 Origin

An improvement oriented heuristic selection strategy combined with a simulated annealing move acceptance as a hyper-heuristic utilising a set of low level constraint oriented neighbourhood heuristics is investigated in 2007 for solving a curriculum based course timetabling problem at Yeditepe University [28]. The low level heuristics are designed similarly to the ones designed in [116], and [175] at which they attempt to improve upon corresponding constraints. The selection method combines a fast reacting greedy and gradient heuristic selection mechanisms.

The performance of the Greedy-gradient - Simulated Annealing Hyper-heuristic (GGHH) has been investigated and compared against a set of three simple selection hyper-heuristics methods, including Simple Random (SR), Greedy (GR) and Choice Function (CF), on a real world problem obtained from the Computer Engineering Department at Yeditepe University and eight problem instances which are randomly generated based

on the definition of the given problem. Table 4.2 summarises the performance of each hyper-heuristic based on 50 runs for each instance as reported in [28]. The evaluation measure *success rate* is used: $s.r.$ = (number of runs for which the perfect solution is obtained)/50. The rankings of the different hyper-heuristics in Table 4.2 are calculated according to the success rates. The lower the ranking, the better a hyper-heuristic is.

TABLE 4.2: Performance ranking of each hyper-heuristic combined with simulated annealing move acceptance over a set of problem instances

label	events	lecturers	GGHH	CF	SR	GR
rp1	200	64	2.5	2.5	2.5	2.5
rp2	200	64	2	2	2	4
rp3	400	128	2	2	2	4
rp4	400	128	1	2	3	4
rp5	800	256	2	2	2	4
rp6	800	256	1	2	3	4
rp7	1600	512	1	2	3	4
rp8	1600	512	1	2	3	4
cse	200	64	2.5	2.5	2.5	2.5
		avr	1.67	2.11	2.56	3.67

The results show that applying different hyper-heuristics (greedy or random gradient) at different stages, in the overall, performs better than simple random, greedy and choice function heuristic selection methods as a part of a selection hyper-heuristic embedding simulated annealing as a move acceptance method. Based on the mentioned results, the approach can be generalised and applied to different problem domains to investigate the performance of applying different hyper-heuristics at different stages of the search.

4.2.2 Methodology

In most of the previous applications of reinforcement learning in hyper-heuristics, a utility value is increased as a reward mechanism and decreased for punishment [19, 176]. It has also been observed in [28] that the memory length affects the performance. The greedy-gradient hyper-heuristic approach somewhat adapts a similar strategy. Instead of a predefined scoring mechanism, the fitness change in between the old and current solution generated after the application of the selected heuristic is used as a utility value. Whenever the utility value of each heuristic is 0, a greedy-like strategy is invoked (Algorithm 12, Lines 1-5). Each heuristic is called one by one using the same solution at hand and the fitness change is recorded as a utility value of the corresponding heuristic.

If a heuristic causes a worsening move, its utility value is set to 0. Then, a heuristic is chosen based on the scores (Algorithm 12, Lines 6 and 7). The *max* function is employed, choosing an option with the highest value and in this case, choosing a heuristic that generates the best improvement. After applying the selected heuristic, its score is updated right away using the fitness change. This strategy neither makes use of a periodic update of scores as in [177], nor forgets the scores as soon as a heuristic is selected as in a greedy method [12]. In the case when one heuristic has a non-zero value, it will be selected as long as the solution improves and the hyper-heuristic will act like a gradient hill climber.

During the heuristic selection process, utility values of a subset of heuristics returned by the *max* function might be the same, necessitating a tie breaking strategy. Two different cases emerge: a non-zero tie score for some heuristics or all zeros. A random selection is performed in the former case. For the latter case, a problem dependent feature is implemented. Another utility array is maintained to keep track of the number of violations due to each constraint type. Again, *max* function is used for determining the highest number of violations and the corresponding constraint type. Hence, the corresponding heuristic is invoked. Then, the utility values of the selected heuristic are updated in both arrays using the new solution.

Algorithm 12: Pseudocode of greedy-gradient heuristic selection method

input: scores, current solution

```
1 if all heuristic scores are 0 then
2 |   invoke each heuristic using the current solution;
3 |   record cost change as the score for each heuristic;
4 |   reset the score of a heuristic to 0 if cost increases;
5 end
6 choose a heuristic based on the scores;
7 in case of a tie, use a tie breaking strategy;
8 return (chosen heuristic id for invocation);
```

Simulated annealing accepts improving moves and non-improving moves with a probability provided in Equation 2.1.

4.3 Dominance-based Random Descent/Gradient Hyper-heuristic with Naïve Move Acceptance (DRD)

In this approach, either a greedy or random descent heuristic selection method is used as a heuristic selection method at any stage. Therefore, each stage will be referred to as greedy or random descent stage depending on the heuristic selection method used. The greedy stage is used to build a *List of Active Heuristics* (LAH) that are expected to perform relatively well. Heuristic selection is followed by a Naïve move acceptance (NV) strategy [60] to decide whether to accept or reject the new solution considering its quality. This approach is initially applied to the first four problem domains of the HyFlex benchmark set and outperformed the default 'mock' eight hyper-heuristics provided by the CHeSC 2011 organisers [170], performing particularly well in the boolean satisfiability and one-dimensional bin-packing problem domains.

The proposed approach is motivated by the idea of dynamically grouping the low level heuristics that are expected to perform relatively well into a list of active heuristics; by (1) reflecting the trade-off between the number of successive steps a low level heuristic is applied and the objective function value achieved; therefore, a heuristic yielding a large improvement in the solution quality after a large number of invocations can be considered as "successful" as one which provides less improvement, but using less invocations. (2) capturing how frequently a heuristic is successful in a given number of steps and feed this information as its selection probability into the second hyper-heuristic.

4.3.1 Methodology

Algorithm 13 provides the pseudocode of the proposed hyper-heuristic. The multi-stage level mechanism starts with a greedy stage. The greedy heuristic selection method allows all the low level heuristics to process a given candidate solution successively for a number of steps to build a *List of Active Heuristics* (LAH) in the greedy stage. LAH is a list of the low level heuristics that are expected to perform relatively well. This is an opposite strategy employed by the Tabu Search based hyper-heuristic [59] which utilises a list to disallow the use of low level heuristics generating worsening results. In the first step of the greedy stage, LAH contains all low level heuristics. The greedy heuristic selection method combined with a dominance-based strategy is used to reduce the number of active heuristics for the next stage. The greedy stage is always followed by a random descent stage. The best solution found during the greedy stage is used as the current

solution to be processed by the random descent stage. In this stage, the random descent heuristic selection method picks a low level heuristic from LAH randomly and applies it to the solution in hand repeatedly until there is no improvement. In the case of obtaining a non-improving solution, the multi-stage level triggers to go into the random descent stage again without accepting the new solution with a probability of P_s; or it will go into the greedy stage for updating the list of active heuristics with a probability of P_u; or it will accept the non-improving solution with a probability of $(1-P_s-P_u)$ and continue with the random descent stage. In [170], and after a set of exhaustive experiments using different combinations of values, P_s and P_u are assigned to 0.50 and 0.25, respectively. The following parts explain how the stages interact in more details.

4.3.1.1 Greedy Stage

In the greedy stage, the greedy heuristic selection method is employed for n successive steps. The best performing heuristics are determined using a strategy inspired from the concept of Pareto Front [178] in multi-objective optimisation. Given a set of k low level heuristic points $LLH=\{LLH_1, LLH_2, \ldots, LLH_k\}$ in 2-dimensional space, each represented by its x (Step) and y (Objective) coordinates. At each step, the objective function value of each solution generated by the corresponding low level heuristic is calculated. Well performing low level heuristics that have the potential to improve within the n steps are those points that are not dominated by any other point. A low level heuristic may make a small improvement in the solution taking a short time and performance-wise this is as good as a heuristic which improves a solution more taking a longer time. Assuming a minimisation problem where we are seeking for the low level heuristics that generate minimum objective function value, a point LLH_i is considered to be dominated by point LLH_j if and only if $(LLH_{i.x} \geq LLH_{j.x})$ and $(LLH_{i.y} \geq LLH_{j.y})$.

Figure 4.2 shows an example on how to build the list of active heuristics for (k=5) low level heuristics. Note that in the Figure, LLH_2 has been added three times to the list and that makes this heuristic to be selected with higher probability. Note also that in Step1, LLH_2 and LLH_3 have been both added to the list, since they have the same objective function value and they are not dominated by any other point.

As there is a limited time to find the best objective function value, the value of n depends on the time of applying the greedy method in one step. n decreases when the time needed to apply the greedy method was high. An exponential function has been considered to find n, where zero is a possible value. However, in case n equals zero, then the list of

Algorithm 13: Pseudocode of the dominance-based and random descent hyper-heuristic

```
1  Let S represent the candidate solution; S_best the best solution; P_s and P_u ∈ [0,1];
2  S ← CreateInitialSolution();
3  S_best ← S;
4  L ← BuildLAH(S_best);
5  LLH ← SelectRandomlyFrom(L);
6  repeat
7  |  S' ← ApplyHeuristic(LLH, S);
8  |  if S' isBetterThan S_best then
9  |  |  S_best ← S';
10 |  end
11 |  if S' isNotBetterThan S then
12 |  |  r ← GenerateUniformRandomNumberIn(0, 1);
13 |  |  if r < P_s then
14 |  |  |  LLH ← SelectRandomlyFrom(L);
15 |  |  |  P_a ← 0;
16 |  |  end
17 |  |  else if r < P_s + P_u then
18 |  |  |  L ← UpdateLAH(S_best);
19 |  |  |  LLH ← SelectRandomlyFrom(L);
20 |  |  |  P_a ← 0;
21 |  |  end
22 |  |  else
23 |  |  |  P_a ← 1;
24 |  |  end
25 |  end
26 |  if S' isBetterThan S then // move acceptance
27 |  |  S = S';
28 |  end
29 |  else if with a probability of P_a% then
30 |  |  S = S';
31 |  end
32 until Exceeded(timeLimit);
33 return S_best;
```

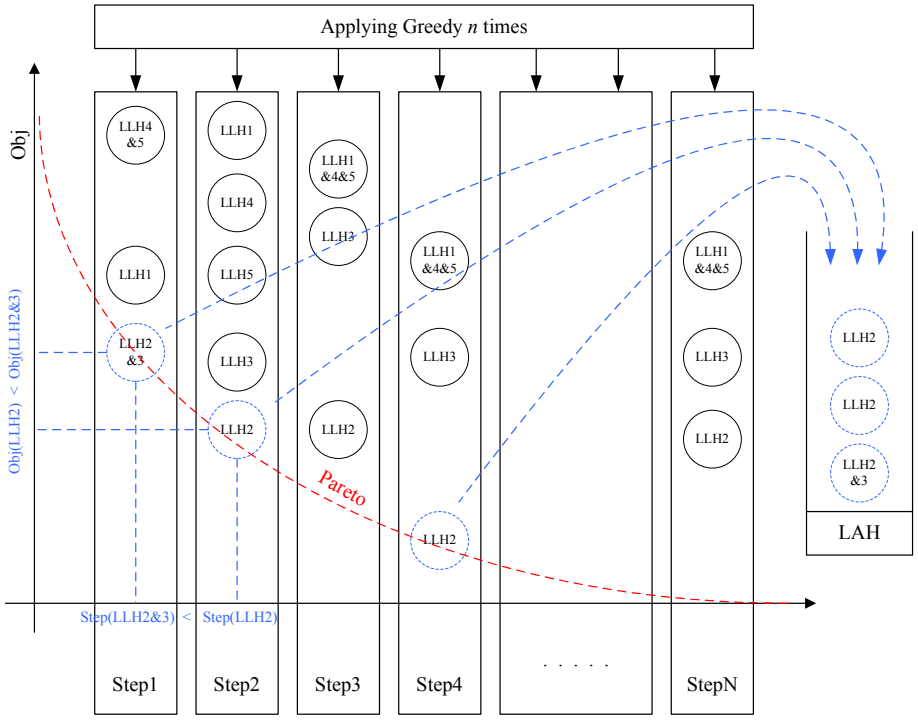


FIGURE 4.2: An illustration showing how the list of active heuristics is built

active heuristics will contain all the k low level heuristics. The value of n is calculated before starting the main loop and it should be an integer value. $n = Ae^{-f(t)}$ where A is the maximum possible value of n. $f(t)$ is the total time of applying the greedy method for one step divided by the limited time that required to complete the whole process; the total time of applying the greedy method for one step equals the summation of the time required to apply each low level heuristic on a given candidate solution. In [170], A was given the value of 9 after a set of trials while applying the method to the HyFlex benchmarks.

4.3.1.2 Random Descent/Gradient Stage

The goal of this stage is to improve the solution at hand as much as possible turning the approach into a hill climber. A low level heuristic is selected randomly from the list of active heuristics, created during the greedy stage, and applied repeatedly until no improvement is achieved.

4.3.1.3 Naïve Move Acceptance

The Naïve move acceptance [60] (Algorithm 13, Lines 26-31) is used as the move acceptance strategy which accepts all improving moves. In case of non-improving moves ($P_a = 1$), the solution accepted with a probability of $(1-P_s-P_u)$; otherwise, the solution remains unchanged ($P_a = 0$).

4.4 Robinhood Hyper-heuristic with an Adaptive Threshold Acceptance (RHH)

This section presents a hyper-heuristic based on a **ro**und-ro**bin** neighbour**hood** (Robinhood) selection mechanism which allocates equal time for each low level heuristic to process a solution in hand. This multi-stage hyper-heuristic approach aims to give a fair chance for each low level heuristic in a selected subset of low level heuristics to execute for a certain duration at a stage. A low level heuristic is chosen in a round robin fashion. Depending on the strategy, the whole set of low level heuristics can be used and the order of low level heuristics can be fixed or varied.

Low level heuristics are classified in two groups: mutation and hill-climbing. The algorithm performs sequences of operations. The sequences are chosen randomly and assigned according to the round robin technique. A sequence is repeated as long as it makes it possible to improve on the quality of the current solution.

Three move acceptance criteria including only improving, improving or equal, and an adaptive acceptance methods are used in this approach. In the adaptive acceptance method, a move that improves the quality of the current solution is always accepted. Deteriorating moves are accepted according to a probability that is adaptively modified at different stages throughout the search.

4.4.1 Methodology

The Robinhood hyper-heuristic (Algorithm 14) is composed of components inspired from previously proposed approaches. The heuristic selection methods presented by Cowling et al. [12] includes *Random Permutation* and *Random Permutation Descent (Gradient)*. This method applies a low level heuristic one at a time sequentially in a randomly generated permutation order. Random Permutation Gradient operates in the same way

with a minor change that is as long as the chosen heuristic makes an improvement in the current solution the same heuristic is employed. Given a time limit of t (Algorithm 14, Lines 2 and 12), and n low level heuristics, the Robinhood hyper-heuristic fixes the number of stages to k and applies all low level heuristics (Line 4) to the current solution in a given order for $t/(n.k)$ time unit at a stage (Line 5).

The proposed hyper-heuristic aims to use all low level heuristics assuming that the domain implementers chose reasonable heuristics which will not be misleading for the search process. Consequently, in the multi-stage level, the low level heuristics are randomly ordered within each group of heuristics: mutational and hill climbing. Inspired by well-known algorithms [53, 56], in which solutions are improved through successive application of mutation and hill climbing, the Robinhood hyper-heuristic uses the same ordering of groups and randomly fixing the ordering of heuristics within each group at a stage. There is also strong empirical evidence in the literature that this ordering is a good choice even for selection hyper-heuristics as reported in [8, 60]. The hyper-heuristic uses the same ordering in the subsequent stage if there is an improvement in the solution quality at a given stage. Otherwise, without changing the group ordering, another random ordering of low level heuristics within each group is generated for the subsequent stage.

Three move acceptance criteria, and hence three hyper-heuristics, are used in this approach. Either only improving, improving or equal, or a modified version of the adaptive acceptance method in [60] is used for the move acceptance. The latter acceptance method accepts all improvements as usual, but the deteriorations are accepted with respect to an adaptively changing rate, denoted as *acceptanceRate*. Assuming a minimisation problem, let $f(x)$ denote the quality of a given solution x, then if $f(S')$ is less than $f(S)$, then S' is accepted, otherwise S' is accepted with a uniform probability of *acceptanceRate* (Algorithm 14, Line 7). Initially, only strictly improving moves are accepted. However, if the solution does not improve for one stage, only the moves generating improving or equal quality new solutions are accepted. If the solution does not improve for another following stage, then threshold move acceptance is activated based on *acceptanceRate*. A reinforcement learning mechanism is used for adapting the value of *acceptanceRate*. If the solution gets stuck at a local optimum for a given stage, then *acceptanceRate* is increased by a δ value for the next stage, making it more likely that a worsening solution is accepted. Conversely, if the solution in hand worsens in a given stage, then the *acceptanceRate* is reduced by δ, making it less likely for a worsening solution to be

accepted. The *acceptanceRate* value updates are intended to help the search navigate out of local optima, and focus the search if it is progressing well.

In [171], the proposed method was shown to work well on the HyFlex benchmark problem domains when $k = 200$ and $\delta = 0.01$, outperforming the mock hyper-heuristics, and taking the fourth place with respect to the twenty approaches in CHeSC 2011.

Algorithm 14: Pseudocode of Robinhood hyper-heuristic

```
1  Initialise();
2  repeat        /* e.g., terminate when the given overall execution time is
   exceeded */
3  |  Update1();      /* set/update relevant parameter/variable values before
   |  entering into a stage or no-op */
4  |  for i = NextLowLevelHeuristicID() do            /* entry of the stage */
5  |  |  while TerminationCriteriaNotSatisfied2() do  /* e.g., terminate when
   |  |  the given time for a heuristic is exceeded */
6  |  |  |  S' ← ApplyLLH(i, S);            /* S and S' are the current and new
   |  |  |  solutions, respectively */
7  |  |  |  MoveAcceptance(S, S');
8  |  |  end
9  |  |  Update2();    /* set/update relevant parameter/variable values after
   |  |  employing a low level heuristic or no-op */
10 |  end
11 |  Update3();     /* set/update relevant parameter/variable values after a
   |  stage or no-op */
12 until TerminationCriterionSatisfied1();
```

4.5 Selection Hyper-heuristic with an Adaptive Threshold Acceptance (HySST)

This section describes the stochastic local search approach of the team HySST (Hyper-heuristic Search Strategies and Timetabling) to high school timetabling which competed in the three rounds of the Third International Timetabling Competition.

We develop and exploit a generalised selective hyper-heuristic. We build on a previous study [8] that demonstrated the effectiveness of a generalised version of the iterated local search approach. Specifically, our hyper-heuristic uses a structured and staged application of multiple perturbative and hill climbing operators as opposed to simple selection from a single pool of all operators.

In [43], a notable difference from standard methods (such as in memetic algorithms) is that the performance is better if the hill climbing is not applied if the mutational operators managed to improve the best solution. We suspected that excessive use of the hill climbing somehow gives over-optimised local solutions that afterwards lead to restricted movement within the search space. If the hyper-heuristics that control the mutational and hill climbing heuristics can be distinguished and implemented separately for this problem domain, an additional improvement could be obtained.

4.5.1 Methodology

Figure 4.3 illustrates how a high level generic single-stage selection hyper-heuristic and HySST multi-stage hyper-heuristic operate. A selection hyper-heuristic in Figure 4.3(a) manages a set of *perturbative* or *constructive* low level heuristics (move operators) [4] and often improves an initially generated solution (s_i) under an iterative process until the termination criterion is satisfied. A generic selection hyper-heuristic does not differentiate between the types of low level heuristics. The multi-stage approach shown in Figure 4.3(b), separates mutational and hill climbing heuristics. Mutational heuristics are employed until some criteria are satisfied, which decide that it is time for intensification, and then a new stage starts employing only hill climbing low level heuristics. The multi-stage level allows switching back and forth between diversification and intensification stages.

The search algorithm is implemented as a time contract algorithm which terminates after a given time, $t_{overall}$ for each instance. The approach consists of an initial solution construction phase followed by an extensive improvement phase using a multi-stage hyper-heuristic. The pseudocode of the algorithm is provided in Algorithm 15. The improvement phase uses the remaining time left ($t_{remaining}$) after the construction of the initial solution which takes t_{init} time.

The multi-stage level divides the search into two main stages: diversification and intensification. Until the given time limit is reached, the proposed approach switches between a diversification stage (stage A) which employs a selection hyper-heuristic combining simple random heuristic selection with an adaptive move acceptance and an intensification stage (stage B) which employs a strict hill climbing process based on local search heuristics. Each stage takes a prefixed amount of time ($t_{MUstage}$ and $t_{HCstage}$). Moreover, stage A controls a small (tunable) threshold value ϵ to relax the degree of consecutive worsening moves during the search process. The threshold acceptance can accept with

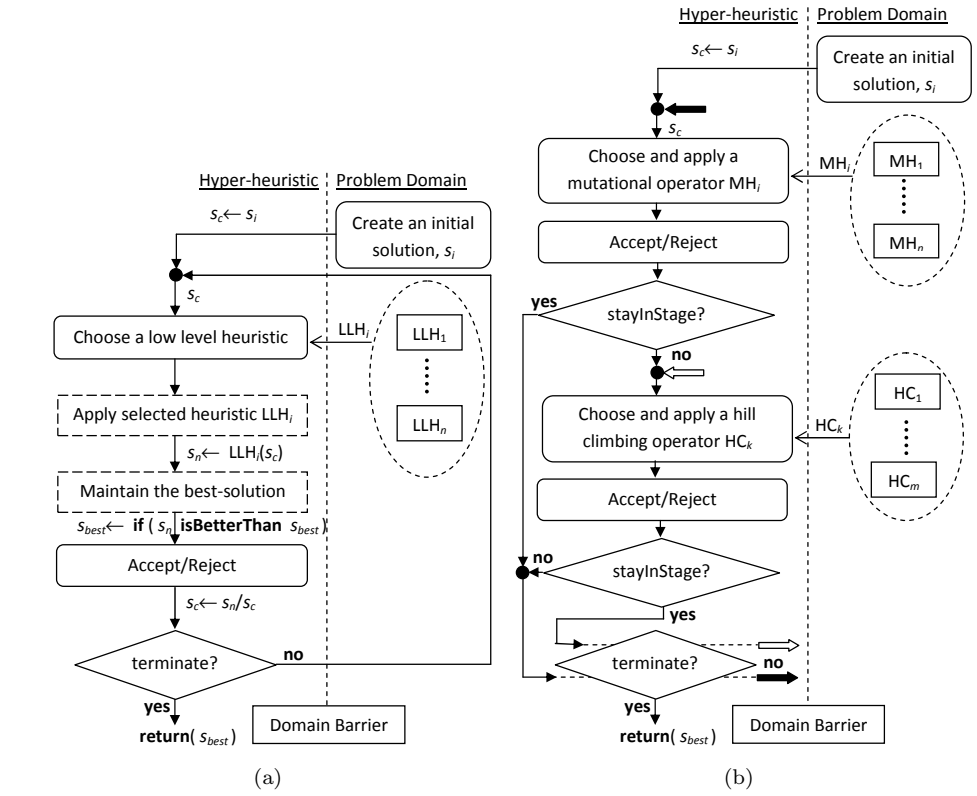


FIGURE 4.3: Illustration of a (a) generic and (b) HySST multi-stage selection hyper-heuristic

factor $(1+\epsilon)$ worse. If no improvement is achieved during a stage, a hill climbing phase is applied using the hill climbing heuristics. A hill climbing step is always non-worsening and so can be repeatedly applied in standard fashion until a local minimum is reached.

The diversification stage makes use of all mutational low level heuristics allowing worsening moves to be accepted via a threshold move acceptance method. The usefulness of restart in randomised search algorithms has already been known and different approaches have been proposed [179, 180]. In this study, we use an adaptive threshold move acceptance method to enable acceptance of worsening moves and partial restarts. The threshold move acceptance method accepts all improved solutions or a worsening solution with a quality better than $(1+\epsilon)$ of the quality of the best solution obtained during the search process at a stage. The acceptance of a worsening solution in this manner could be considered as a partial restart on a given solution. The degree of a partial restart is indicated by *level* controlling the threshold value of ϵ. The larger the threshold is, the lower the quality of solutions that get accepted. The diversification stage is repeated within the time limits as long as the best solution obtained at the end

Algorithm 15: Pseudocode of the HySST multi-stage hyper-heuristic

```
1  S ← CreateInitialSolution(); // takes t_init time
2  t_remaining ← t_overall − t_init;
3  S_best ← S;
4  thresholdList[] ← {ϵ_1, ϵ_2, ..., ϵ_maxLevel};
5  level ← 1;
6  repeat
7      S_beststage ← S;
8      S_startstage ← S;
9      ϵ ← thresholdList[level];
10     while notExceeded(t_MUstage && t_remaining) do // stage A entry using ϵ
11         LLH ← SelectRandomlyFrom(MutationalHeuristics);
12         S' ← ApplyHeuristic(LLH, S);
13         if S' isBetterThan S_best then
14             S_best ← S';
15         end
16         if S' isBetterThan S_beststage then
17             S_beststage ← S';
18         end
19         S ← MoveAcceptance(S, S', S_beststage, ϵ); // threshold acceptance
20     end
21     if S_beststage isNotBetterThan S_startstage then
22         while notExceeded(t_HCstage && t_remaining) do // stage B entry
23             LLH ← SelectRandomlyFrom(HillClimbers);
24             S'' ← ApplyHeuristic(LLH, S);
25             if S'' isBetterThan S_best then
26                 S_best ← S'';
27             end
28             if S'' isBetterThan S_beststage then
29                 S_beststage ← S'';
30             end
31             S ← S''; // accept all moves
32         end
33     end
34     if S_beststage isNotBetterThan S_startstage then
35         if level == maxLevel then
36             S ← S_startstage;
37             level ← 1;
38         end
39         else
40             level++;
41         end
42     end
43 until Exceeded(t_remaining);
44 return S_best;
```

of a stage ($S_{beststage}$) is of better quality than the best solution in hand at the start of a stage ($S_{startstage}$). In a way, the diversification stage is parametrised depending on ϵ. Each diversification stage using a different ϵ is considered as a different stage. If a diversification stage produces a worsening resultant solution, then the intensification stage which makes use of hill climbing heuristics kicks in. If a solution cannot be improved even after an intensification stage, the ϵ value is increased to allow even larger changes in the solution causing larger worsening in its quality in the stage. We have used a discrete choice for the ϵ values and grabbed the next (previous) item from an ordered fixed-size threshold list in order to increase (decrease) its value. The minimum and maximum threshold values are limited with the first and last items in the list.

4.6 Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance (DRW)

In this approach, two selection hyper-heuristics are combined by employing them successively in a structured and staged manner. The approach extends the hyper-heuristics described in the previous sections. The dominance-based method combined with the round-robin strategy explained in Sections 4.3 and 4.4 are used to determine the list of active heuristics. The threshold move acceptance method explained in Section 4.5 is improved by incorporating a logarithmic equation.

4.6.1 Methodology

The pseudocode of the approach is in Algorithm 16. Lines 6-23 and Lines 25-26 illustrate the first and second hyper-heuristics, respectively. The first hyper-heuristic randomly selects a low level heuristic from an active pool of heuristics, denoted as LLH in a score proportionate manner using a roulette wheel strategy (Line 6). If $score_i$ is the score of the i^{th} heuristic, then the probability of a heuristic being selected is $score_i / \sum_{\forall j}(score_j)$. Then the selected heuristic is applied to the solution in hand (Line 7). Initially, each heuristic has a score of 1, making the selection probability of each heuristic equally likely. The first hyper-heuristic always maintains the best solution found so far, denoted as S_{best} (Lines 10-12) and keeps track of the time since the last improvement. The move acceptance component of this hyper-heuristic (Lines 8-19) is a threshold acceptance method controlled by a parameter, ϵ, accepting all improving moves. A non-improving solution is accepted only if the quality is better than $(1 + \epsilon)$ of the quality of the best

solution obtained (Line 16). Whenever the best solution can no longer be improved for a complete $timeLimit_2$ second (Line 20), ϵ gets updated (Line 22) according to Equation 4.1.

$$\text{UpdateEpsilon}(x) = \frac{\lceil \log(x) \rceil + rand(1, \lceil \log(x) \rceil)}{x} \tag{4.1}$$

where $x = f(S_{best})$ which is the objective value of the best solution obtained and $rand(lb, ub)$ returns a random integer in $[lb, ub]$. If $f(S_{best})$ is 0, the algorithm terminates and so this case is not considered in the threshold update.

This novel move acceptance method operates in an unusual way while dealing with non-improving moves. After ϵ gets updated, during the initial iterations of the search process, moves *slightly* worse than the best solution which is achieved right before the update are accepted. At a later stage after the update, if a new best solution is obtained, the method relaxes the bound on the objective value of worsening solutions further and starts accepting the ones with larger changes in the objective value.

The second hyper-heuristic dynamically starts operating (Lines 24-27) whenever there is no improvement in the quality of the solution for $timeLimit_3$ second (Line 24). It determines the active pool of heuristics (LLH) from the full set of low level heuristics, denoted as LLH_{all} will be used in the following stage extending the idea of a dominance-based heuristic selection as introduced in Section 4.3 and adjusts the score of each low level heuristic dynamically. Firstly, ϵ is updated in the same manner as in the first hyper-heuristic and never gets changed during this phase. Then a greedy strategy is employed using all low level heuristics for a certain number of steps, which is fixed to the number of low level heuristics. Step by step, this hyper-heuristic builds a set of solutions associating each with the low level heuristic producing that solution reflecting the trade-off between the objective achieved by each low level heuristic and the number of steps involved. At the end of this phase, a pareto front is obtained using the non-dominated solutions from the whole set. The low level heuristics on the pareto front are used to form the active pool of low level heuristics. If more than one low level heuristic generates the same objective value which ends up on the pareto front, they all get to enter into this pool. The number of occurrences of each low level heuristic is assigned as its score to be used in the first hyper-heuristic.

At each step, each low level heuristic is applied to the same input solution for a fixed time τ in a round-robin fashion (Section 4.4) while considering the threshold move acceptance method. Some of the low level heuristics may take more time than the others and therefore, we used the round-robin approach in order to treat all the low

Algorithm 16: Pseudocode of the dominance-based and roulette wheel hyper-heuristic

```
1  Let LLH_all = {LLH_1, LLH_2, ..., LLH_M} represent set of all low level heuristics with
   each heuristic being associated with a score, initially set to 1;
2  Let S_best represent the best schedule;
3  S ← S_i; S_best ← S_i; LLH ← LLH_all;
4  repeat
5      heuristicID ← SelectLowLevelHeuristic(LLH);
6      S' ← ApplyHeuristic(LLH_heuristicID, S);
7      if f(S') < f(S) then
8          S ← S';
9          if f(S') < f(S_best) then
10             S_best ← S';
11         end
12     end
13     else
14         UpdateHeuristicSetting(heuristicID);
15         if f(S') < (1 + ε)f(S_best) then
16             S ← S';
17         end
18     end
19     if noImprovement(timeLimit_2) then
20         S ← S_best;
21         ε ← UpdateEpsilon(f(S_best));
22     end
23     if noImprovement(timeLimit_3) then
24         ε ← UpdateEpsilon(f(S_best));
25         S, LLH ← DecideLowLevelHeuristics(S_best, LLH_all, timeLimit_1);
26     end
27 until Exceeded(timeLimit_1);
28 return S_best;
```

level heuristics equally. In [173], for example, τ is assigned to $5n/q$, n/q and 1 iterations for each heuristic in LLH_{small}, LLH_{medium} and LLH_{large}, respectively, where n is the number of activities and q is the number of projects. If a low level heuristic produces a solution identical to the input, that invocation is ignored. Otherwise, the objective of the new solution together with the low level heuristic which produced that solution gets recorded. If all heuristics cannot generate a new solution, then they are reconsidered all together. Once all heuristics are applied to the input and gets processed for the step, the best solution propagates as input to the next greedy step. If the overall given time limit ($timeLimit_1$) is exceeded, then the second hyper-heuristic terminates before completing through all steps and uses the solution set in hand.

Figure 4.4 illustrates a run of the second hyper-heuristic with four low level heuristics ($LLH_{all} = \{LLH_1, LLH_2, LLH_3, LLH_4\}$). Assuming that the pareto front contains 3 points. The first, second and third points on the front are associated with $\{LLH_1, LLH_2\}$, $\{LLH_1\}$ and $\{LLH_1, LLH_3\}$, respectively. Hence, in the next stage, the first hyper-heuristic ignores the fourth low level heuristic ($LLH = \{LLH_1, LLH_2, LLH_3\}$) and scores of LLH_1, LLH_2 and LLH_3 are assigned to 3, 1 and 1, respectively. Hence, selection probability of LLH_1 becomes 60%, while it is 20% for LLH_2 and LLH_3.

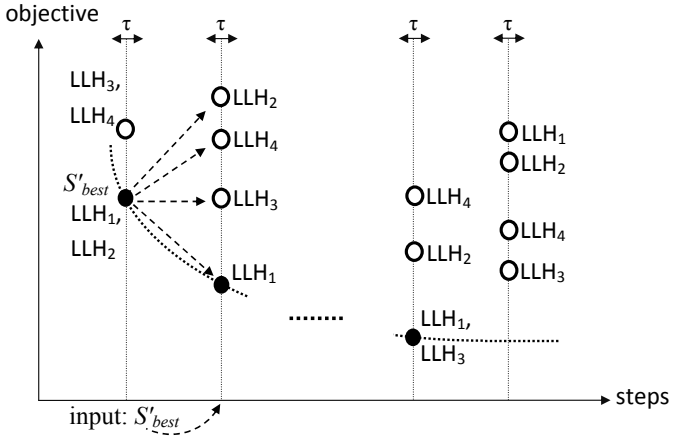


FIGURE 4.4: An illustation of how the second greedy hyper-heuristic operates

4.7 Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH)

This section introduces a multi-stage hyper-heuristic utilising two hyper-heuristics and controlled by the multi-stage level as provided in Algorithms 17, 18 and 19. In one stage, a subset of "useful" low level heuristics, each associated with a score is determined by a hyper-heuristic embedding a greedy heuristic selection method (Algorithm 17, Lines 12-28). Only that subset of low level heuristics is then used in the other stage (Algorithm 17, Lines 9-11) and at each step, a heuristic is selected using a roulette wheel strategy based on those scores. As a move acceptance component of the multi-stage hyper-heuristic, we extend the threshold move acceptance method explained in Section 4.6 and use it in both stages (Algorithm 18, Line 5 and Algorithm 19, Line 6), however the threshold values are treated in a different way in each stage as explained in this section. Additionally,

following the previous approach in Section 4.4, any selected low level heuristic is executed for a certain duration, τ (Algorithm 18, Lines 3-7 and Algorithm 19, Lines 4-8).

In this approach, we assume that a number of low level heuristics for a given problem domain are already provided. We form "new" heuristics by pairing up each low level heuristic and invoking them successively. Consequently, given n heuristics, we end up with $(n+n^2)$ low level heuristics in the overall. The technique of combining two heuristics is also known as *relay hybridisation* [27] which applies the second low level heuristic to the solution generated by the preceding low level heuristic. The motivation behind relay hybridisation is that, although a low level heuristic that does not generate any improvement, it may still be useful when used in combination with another low level heuristic.

The relevant parameter (e.g. intensity or depth of the search) setting of any selected low level heuristic gets updated to a random value in case the move does not improve the candidate solution, otherwise the same setting is kept.

4.7.1 Stage One Hyper-heuristic

In stage one (S1HH) (Algorithm 18), the roulette wheel selection based hyper-heuristic chooses and applies randomly a low level heuristic based on a score associated with each low level heuristic (Algorithm 18, Lines 2 and 4). Given n LLHs, LLH_{n+1} denotes the heuristic produced after relay hybridisation of the pair LLH_1+LLH_1, LLH_{n+2} of the pair LLH_1+LLH_2, and so on. Assuming that the i^{th} low level heuristic LLH_i has a score of $score_i$, then the probability of that heuristic being selected is $score_i / \sum_k (score_k)$. Initially, all single heuristics are assigned a score of 1, while the rest of the paired heuristics are assigned to a score of 0. The stage one hyper-heuristic always maintains the best solution found during the search process, denoted as $S_{beststage}$ and keeps track of the time since the last improvement (Algorithm 18, Line 6). The move acceptance approach directly accepts improving moves, while non-improving moves are accepted if the objective function value of the candidate solution is better than $(1 + \epsilon)$ of the objective function value of the best solution obtained in the relevant stage (Algorithm 18, Line 5). Whenever the best solution during a stage can no longer be improved for a duration of d, ϵ gets updated according to Equation 4.2.

$$\epsilon = \frac{\lfloor \log(f(S_{beststage})) \rfloor + c_i}{f(S_{beststage})} \tag{4.2}$$

Algorithm 17: *MultiStageLevel*

$LLH_{all} \leftarrow \{LLH_1, ..., LLH_n, LLH_{n+1}, ..., LLH_{n+n^2}\}$. Each LLH_i is associated with a $score_i$;
Let $S_{current}$ represent the candidate (current) solution; $S_{inputstage1}$ the input solution to S1HH; $S_{inputstage2}$ the input solution to S2HH; $S_{bestoverall}$ the best solution obtained so far; $S_{beststage}$ the best solution obtained in the relevant stage;
Let P_{S2HH} represent the probability to apply S2HH;
Let $f(x)$ represent the objective value of a solution x for a given minimisation problem;
$S_{current}, S_{inputstage1}, S_{inputstage2}, S_{bestoverall}, S_{beststage} \leftarrow S_{initial}$;
$score_{all} \leftarrow \{1, ..., 1, 0, ..., 0\}$; $counter \leftarrow 0$;
Let C be the set of threshold values to be used by the move acceptance;
while `notSatisfied`*(terminationCriterion)* **do**
    while `notSatisfied`*(stageOneTerminationCriteria)* **do**
        $S_{current}, S_{bestoverall}, S_{beststage} \leftarrow$ `ApplyStageOneHH`$(LLH_{all}, score_{all}, S_{inputstage1}, S_{bestoverall}, c_{counter})$;
    end
    if $Random(0,1) < P_{S2HH}$ **then**
        `// Pre-processing steps of S2HH`
        $S_{inputstage2} \leftarrow S_{beststage}$;
        if $f(S_{beststage}) \geq f(S_{inputstage2})$ **then**
            if $counter = (|C| - 1)$ **then**
                $S_{inputstage2} \leftarrow S_{current}$;
            end
            $counter \leftarrow (counter + 1)\ mod\ |C|$;
        end
        else
            $counter \leftarrow 0$;
        end
        while `notSatisfied`*(stageTwoTerminationCriteria)* **do**
            $S_{bestoverall}, S_{beststage}, S_{beststep}, paretoArchieve \leftarrow$ `ApplyStageTwoHH`$(LLH_{all}, S_{inputstage2}, S_{bestoverall}, c_{counter})$;
            $S_{inputstage2} \leftarrow S_{beststep}$
        end
        `// Post-processing steps of S2HH`
        $score_{all} \leftarrow$ `computeScoresBasedOnDominance`$(paretoArchieve)$;
    end
    else
        $score_{all} \leftarrow \{1, ..., 1, 0, ..., 0\}$;
    end
    $S_{inputstage1} \leftarrow S_{beststage}$;
end
return $S_{bestoverall}$;

Algorithm 18: *ApplyStageOneHH*

input : $LLH_{all}, score_{all}, S_{inputstage1}, S_{bestoverall}, c_{counter}$
output: $S_{current}, S_{bestoverall}, S_{beststage}$

1 $S_{current}, S_{beststage} \leftarrow S_{inputstage1}$;
2 $hIndex \leftarrow$ `rouletteWheelSelection` $(LLH_{all}, score_{all})$;
3 **while** `notExceeded`(τ) & `notExceeded`$(timeLimit)$ **do**
4 $\quad S_{new} \leftarrow$ `applyHeuristic`$(LLH_{hIndex}, S_{current})$;
5 $\quad S_{current} \leftarrow$ `moveAcceptance`$(S_{current}, S_{new}, c_{counter})$;
6 $\quad S_{beststage}, S_{bestoverall} \leftarrow$ `updateBestValues`$(S_{current})$;
7 **end**
8 **return** $S_{current}, S_{bestoverall}, S_{beststage}$;

where $f(S_{beststage})$ is the objective value of the best solution obtained during the stage and c_i is an integer value in C=$\{c_0, ..., c_i, ..., c_{(k-1)}\}$, where $c_{(i-1)} < c_i$ for $0 < i < k$ and $k = |C|$. If $f(S_{beststage})$ is less than 1, ϵ takes a small value ~ 0.

The value of c_i never changes in this stage but it might get updated in stage two as explained in the following section. In the first execution of stage one, c_0 is used by default.

If the overall given time limit ($timeLimit$) is exceeded, or there is no improvement in the quality of the best solution obtained during the stage for a duration of s_1, then the stage one hyper-heuristic terminates (Algorithm 17, Line 9).

4.7.2 Stage Two Hyper-heuristic

Algorithm 19: *ApplyStageTwoHH*

input : $LLH_{all}, S_{inputstage2}, S_{bestoverall}, c_{counter}$
output: $S_{bestoverall}, S_{beststage}, S_{beststep}, paretoArchieve$

1 $S_{beststage} \leftarrow S_{inputstage2}$;
2 **for** $i = 0; i < (n + n^2); i++$ **do**
3 $\quad S_{current} \leftarrow S_{inputstage2}$;
4 $\quad$ **while** `notExceeded`(τ) & `notExceeded`$(timeLimit)$ **do**
5 $\quad\quad S_{new} \leftarrow$ `applyHeuristic`$(LLH_i, S_{current})$;
6 $\quad\quad S_{current} \leftarrow$ `thresholdAcceptance`$(S_{current}, S_{new}, c_{counter})$;
7 $\quad\quad S_{beststage}, S_{bestoverall}, S_{beststep}, heur_{beststep} \leftarrow$ `updateBestValues`$(S_{current})$;
8 $\quad$ **end**
9 $\quad paretoArchieve \leftarrow$ `update`$(S_{beststep}, heur_{beststep})$;
10 **end**
11 **return** $S_{bestoverall}, S_{beststage}, S_{beststep}, paretoArchieve$;

The aim of this stage (S2HH) (Algorithm 19) is to reduce the set of low level heuristics and adjust their scores according to their "performance" using the idea of the dominance-based heuristic selection (Section 4.6). A score of 0 indicates that the corresponding heuristic will not be used in the following stage. The reduced set of low level heuristics along with the associated score are fed into the stage one hyper-heuristic.

Firstly, ϵ is set using Equation 4.2 for once at the start of this stage. Having a sorted *circular* list of values $C=\{c_0, ..., c_i, ..., c_{(k-1)}\}$ enables adaptive control of the level of diversification and gives flexibility of relaxing the threshold further whenever necessary allowing larger worsening moves. Initially, c_i takes the value of c_0. If the best solution obtained after applying stage one does not improve for a stage and stage two hyper-heuristic is applied, the parameter takes the next value on the list, that is, for example, c_1, allowing a larger worsening move to be accepted, and so on. If stage one hyper-heuristic manages to improve the solution and then stage two hyper-heuristic is gets applied, the parameter is reset to c_0. By default, $S_{beststage}$ is fed as an input to the next stage. There might be a case when even the $(c_{(k-1)})$ value is not sufficient to escape from a local optimum and the current (candidate) solution is worse than $S_{beststage}$. If the second stage gets executed at this point of the search process, then the current solution is fed into the next stage as input to allow further diversification (Algorithm 17, Lines 13-22). It is possible for a given problem domain that Equation 4.2 could return a value of 0, then c_i is assigned to one of the values in C at random. After c_i is updated, it does not get changed during the execution of the remaining steps of this stage.

A greedy hyper-heuristic is applied using LLH_{all} (Algorithm 19, Lines 2-10) for a fixed number of steps s_2. At each step, all the objective function values obtained by applying all the low level heuristics are recorded only if they generate solutions different in quality to the input solution. If all heuristics cannot generate a new solution, then they considered all to have the worst possible objective value. The greedy approach takes the best generated solution obtained at a step and feeds it as an input solution to the next step.

At the end of the stage, the non-dominated solutions each associated with the low level heuristic that generated them are determined from the archive (Algorithm 17, Line 27). Then the score of each "non-dominated" low level heuristic is increased by 1. It is potentially possible that a low level heuristic could produce a non-dominated solution more than once and so get a higher score indicating the frequency of such success. In the case of a tie where multiple low level heuristics produce the same non-dominated solution for a given step, their scores are all incremented.

Due to the costly run time that the second stage hyper-heuristic introduces, taking $s_2.(n + n^2)$ steps in the overall, a relatively low value for s_2 is preferred. Additionally, we have introduced a probability parameter (P_{S2HH}) (Algorithm 17, Line 12) in order to limit the use of this stage often. The stage terminates if s_2 steps are fully executed or overall given time limit (*timeLimit*) is exceeded (Algorithm 17, Line 23).

Figure 4.5 provides an example of how the stage two hyper-heuristic operates for $n = 2$ low level heuristics (LLH_{all} = {LLH_1, LLH_2, LLH_3=$LLH_1 + LLH_1$, LLH_4=$LLH_1 + LLH_2$, LLH_5=$LLH_2 + LLH_1$, LLH_6=$LLH_2 + LLH_2$}) and $s_2 = 4$ steps. The low level heuristics on the pareto front are {LLH_1, LLH_2}, {LLH_1} and {LLH_3}. Hence, the scores of the fourth, fifth and sixth low level heuristics are zero; and the scores of LLH_1, LLH_2 and LLH_3 are assigned to 2, 1 and 1, respectively. Therefore, the probability of selecting LLH_1 in stage one hyper-heuristic becomes 50%, while it is 25% for LLH_2 and LLH_3.

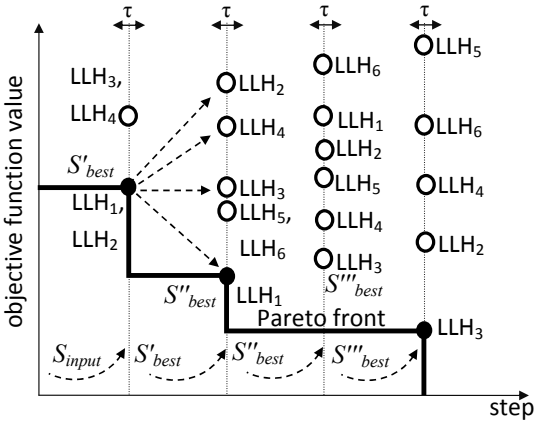


FIGURE 4.5: An example of how the stage two hyper-heuristic works. Note that at step1: LLH_5 and LLH_6 generate solutions with the same quality as in S_{input}

4.8 Summary

A selection hyper-heuristic is a general-purpose search methodology that mixes and controls a given set of heuristics for solving a computationally hard problem. Such high level methods do not require any modification while being applied to a new (unseen) problem domain. Moreover, the component-based design of selection hyper-heuristics enables re-usability of those components as well. Up to this date, most of the selection hyper-heuristics performing single point based search contain two key components: heuristic selection and move acceptance. This chapter presents one of the initial studies

that explicitly addresses a way to combine multiple hyper-heuristics in a staged-manner. A general multi-stage hyper-heuristic framework is presented and its main components are described. This framework is used as a basis to implement several multi-stage hyper-heuristics with synergistic components embedding several hyper-heuristics.

Kalender et al. [28, 29] applied a multi-stage hyper-heuristic to curriculum-based university course timetabling. The proposed approach uses the simulated annealing move acceptance with two combined hyper-heuristics including Greedy and Random Gradient hyper-heuristics (GGHH). A multi-stage hyper-heuristic which combines *Dominance-based heuristic selection*, aims to detect the active low level heuristics, and the Random Descent hyper-heuristic (DRD) is proposed in Section 4.3. A multi-stage hyper-heuristic known as *Robinhood* hyper-heuristic (RHH) combining round-robin strategy-based neighbourhood selection and three move acceptance methods is explained in Section 4.4. The selection heuristic method applies the mutational heuristics on the candidate solution, then hill climbing heuristics and assigns equal time for each low level heuristic. Three move acceptance criteria including only improving, improving or equal, and an adaptive acceptance methods are used in RHH. A multi-stage hyper-heuristic named Selection Hyper-heuristic with an Adaptive Threshold Acceptance (HySST) combines two hyper-heuristics, Simple Random with Adaptive Threshold move acceptance and Simple Random with Accept All Moves acceptance methods is described in Section 4.5. The first hyper-heuristic in this approach is applied on mutational operators, and the second is applied on hill-climbers. A multi-stage hyper-heuristic which combines two hyper-heuristics, Dominance-based hyper-heuristic and Roulette Wheel selection with Adaptive Threshold move acceptance (DRW) is proposed in Section 4.6. This approach makes use of DRD, HySST and RHH approaches. Following that some of the low level heuristics in Robinhood hyper-heuristic may almost become useless at different stages of the search process, then by reducing the number of heuristics involved in the search process at a stage by incorporating the dominance-based method may improve the performance further. The last multi-stage hyper-heuristic named Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) extends the DRW approach and makes use of the *relay hybridisation* [27] technique which applies a low level heuristic to a solution generated by applying a preceding heuristic. One of the hyper-heuristics aims to reduce the number of low level heuristics discovering the "potentially" useful ones at a given stage during the search process and adjust the probability of each low level heuristic being selected in the following stages. This hyper-heuristic extends the low level heuristic set first by creating "new" heuristics through relay hybridisation, and then a dominance based

learning strategy is employed reducing the number of heuristics. This strategy captures the trade-off between the extent of improvement that a heuristic can generate and the number of steps it takes to achieve that improvement. Moreover, each chosen low level heuristic in the "reduced" set is associated with an adaptively decided selection probability to be used in the following stages. The second hyper-heuristic mixes the "reduced" set of low level heuristics with the given probabilities during the search process.

We have provided an overview of some combinatorial optimisation problems and previous approaches used for solving those problems, then discussed the initialisation method and low level heuristics that can be used by selection hyper-heuristics for solving those problems. This chapter has introduced a set of multi-stage hyper-heuristics combining multiple selection hyper-heuristics under a proposed single point based search framework. The following chapter provides the empirical results and analysis from applying the proposed multi-stage hyper-heuristics to a variety of combinatorial optimisation problems and their performances with respect to the state-of-the-art hyper-heuristics or other search methodologies.

Chapter 5

State-of-the-art in Problem Solving and Multi-stage Hyper-heuristics

It is always of interest to researchers and practitioners to know the state-of-the-art approach for solving a specific problem. Although hyper-heuristic research aims for the level of generality, still knowing the relative position of hyper-heuristics with respect to the state-of-the-art for a specific domain would be useful. Another curiosity is whether a hyper-heuristic can outperform all other methods and become state-of-the-art for a specific domain while still being general. One way of establishing state-of-the-art for a specific problem is through competitions/challenges. Therefore, we have actively joined two international competitions, ITC 2011 and MISTA 2013. This chapter summarises the results of the team HySST (Hyper-heuristic Search Strategies and Timetabling) to high school timetabling which competed in all three rounds of the Third International Timetabling Competition ITC 2011. HySST multi-stage hyper-heuristic generated the best new solutions for three given instances in Round 1 and gained the second place in Rounds 2 and 3. This chapter also presents the results of the winning approach competed in MISTA 2013 challenge for the multi-mode resource-constrained multi-project scheduling problem. The approach combines hybrid approaches, and uses the Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance (DRW) multi-stage hyper-heuristic method with the ability to exploit the computing power of multi-core machines. The proposed algorithm significantly outperformed the other approaches

during the competition producing the best solution for 17 out of the 20 test instances and performing the best in around 90% of all the trials.

The Java implementation of the HyFlex interface was used in a cross-domain heuristic challenge (CHeSC 2011). The results form this competition and six problem domain implementations became a benchmark in selection hyper-heuristic studies. The winning state-of-the-art approach is an elaborate but complicated algorithm[1] which makes use of machine learning techniques [27]. The proposed multi-stage hyper-heuristics presented in Chapter 4 are tested on HyFlex to examine their level of generality. The Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) outperforms the competing hyper-heuristics in CHeSC 2011. Moreover, the MSHH is further tested on the magic square problem domain.

5.1 HySST in an International Timetabling Competition

HySST multi-stage hyper-heuristic has been tested on real world instances as a solver for the third International Timetabling Competition (ITC 2011). This approach proved an excellent performance compared to other approaches; in round 1 it generated the best solution for three given instances, and took the second place of the competition for rounds 2 and 3 [43].

At the end of the competition, there were four additional teams who were able to submit solutions for ITC 2011: GOAL, HFT, Lectio and VAGOS. The teams HFT, Lectio and VAGOS attempted to develop tailored solutions in the given the limited time, while the HySST team preferred applying a multi-stage hyper-heuristic. Ultimately, our approach performed better than the approaches proposed by those teams, though could not beat the approach proposed by GOAL. Moreover, Lectio, GOAL and HFT reported that they spent more than hundred days on the project while VAGOS and HySST spent between ten to fifty days (see the competition website for more details). This is an indication of how fast the algorithm was implemented with less effort and expertise in the area, considering that Lectio is a team formed of members from a company and GOAL had expertise in the area of high school timetabling, as they provided the Brazilian instances. However, the primary point of the work is that it shows the utility of the multi-stage hyper-heuristic in that it makes better usage of the domain specific

[1] the publicly available implementation of the winning algorithm counts over 3000 lines of code and introduces over 45 parameters

heuristics, and in particular demonstrates the advantages of multi-stage methods with adaptive relaxations. Note that most of the multi-stage hyper-heuristics described in Chapter 4 were not proposed by the time of the ITC 2011 competition. Here, we present the performance analysis and the results of HySST in ITC 2011.

5.1.1 Performance Analysis

The HySST multi-stage stochastic local search hyper-heuristic managing all low level heuristics and using the adaptive move acceptance for partial restarts turned out to be very effective in solving high school timetabling problems. If a small value of ϵ does not provide any improvement in the solution quality in stage A, then its value is increased which causes acceptance of lower quality solutions and escape from a local optimum. Figure 5.1 provides a sample run on Instance4-Brazil using ϵ ={0.001, 0.33, 1.99} ignoring the strict hill climbing process in stage B. The plot shows that the *reheats* do lead to drastic drops in the cost of a solution, and without the reheats, the search is clearly stuck. For example, the stage indicated as level 1 in Figure 5.1 (red points) performing almost strict hill climbing based on $\epsilon = 0.001$ is eventually stuck and even the stage indicated as level 2 in which ϵ is 0.33 (green points) gets stuck. The blue points (level 3) in Figure 5.1 are a strong relaxation where $\epsilon = 1.99$, but do lead to later improvements.

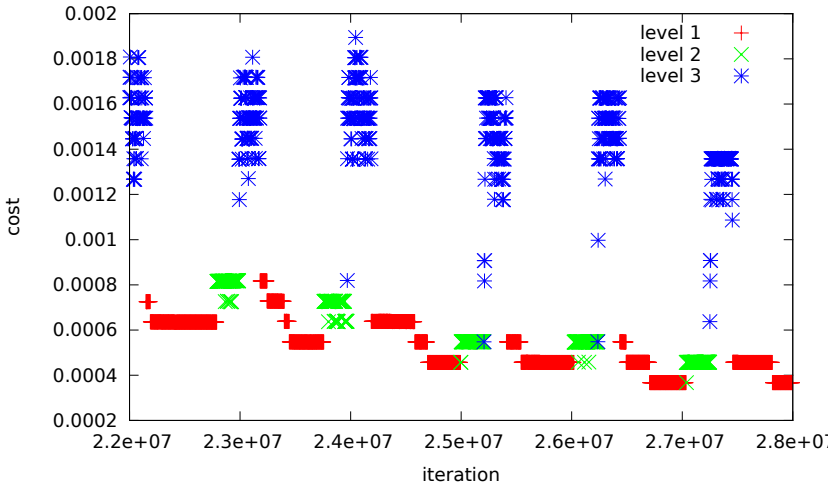


FIGURE 5.1: Cost versus iteration plot of a sample run towards the end of the search process which is obtained by applying the proposed approach to Instance4-Brazil using the threshold list of {0.001, 0.33, 1.99}

The hill climbing algorithms fail to produce an improving solution most of the time and for most of the instances. For example, Figure 5.2 displays a sample run where hill climbing yields no improvement in any stage. Yet, it has been observed that the stage B based on hill climbing is useful for achieving high quality solutions; in particular, for the Australian high school timetabling instances. At the end of a run on those instances, the proposed approach using hill climbing yields (even if slightly) better results than the one which does not use hill climbing. Figure 5.3 shows a sample run on the BGHS98 instance with and without hill climbing (stage B). After the mutational heuristics are employed at a stage A, regardless of the threshold level, hill climbing generates a non-worsening feasible solution. Unfortunately, this seems to occur once and no more improvement could be achieved via hill climbing algorithms.

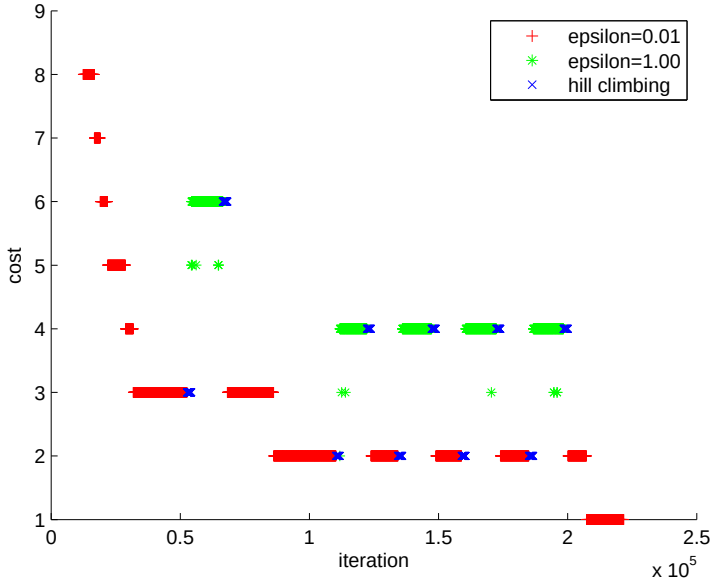
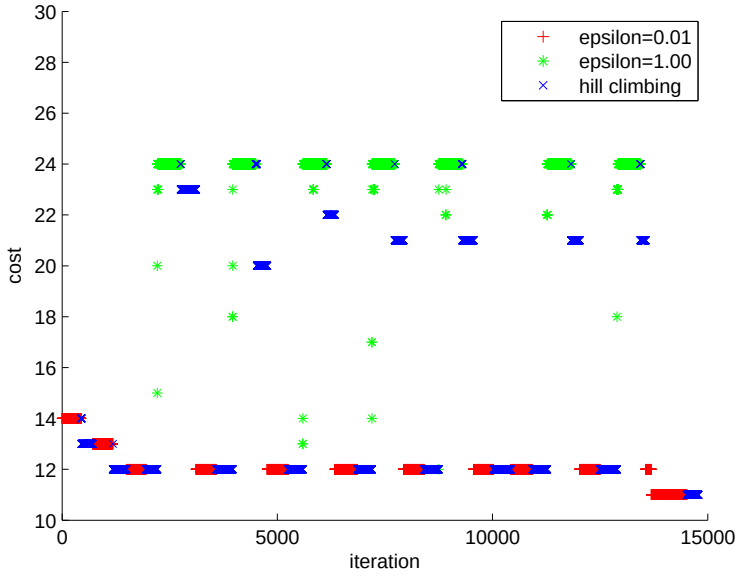


FIGURE 5.2: Cost versus iteration plot of a sample run on WesternGreeceUni5

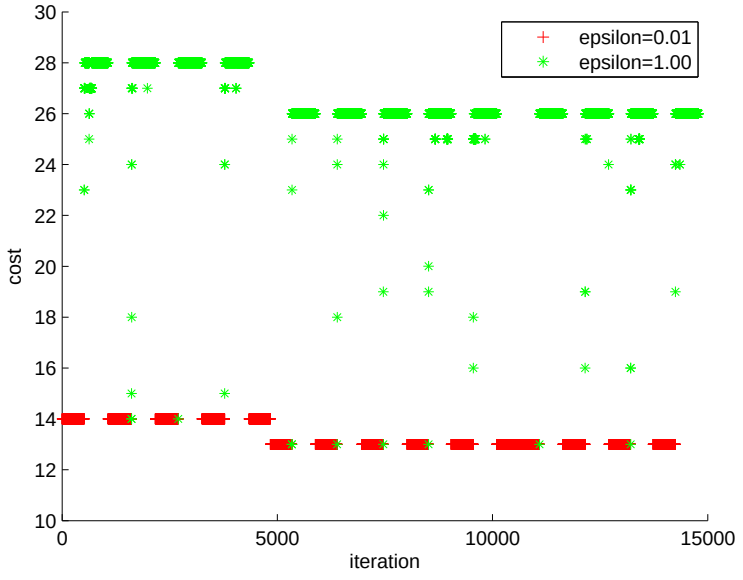
5.1.2 ITC 2011 Competition Results

The HySST approach successfully improved upon the best previously known solutions (BKNs) for the Australian high school timetabling instances of BGHS98, SAHS96 and TES99 in the first round of the competition as shown in Table 5.1.

Table 5.2 summarises the results of Round 2 on the hidden instances. The column labelled as "KHE" shows the average quality of ten initial solutions produced by the



(a)



(b)

FIGURE 5.3: Cost versus iteration plot for BGHS98 - Australia (a) with and (b) without hill climbing (stage B)

TABLE 5.1: The performance of the HySST approach in Round 1. The quality (cost) of a solution is indicated as feasibility-value.objective-value and BKN is the best previously known solution

Dataset	BKN	HySST
BGHS98 - Australia	7.433	3.494
SAHS96 - Australia	23.044	8.052
TES99 - Australia	26.134	1.140

constructive approach of the KHE library. The best feasibility/objective values over ten runs for each instance show that HySST performs the best on two instances of Kottenpark2003 and Kottenpark2005A from the Netherlands and worst on Instance1 - Kosovo. The results reveal that HFT and Lectio did not use the default constructive approach and they obtained solutions of quality which are even worse than the constructive approach achieves for 16 and 6 instances, respectively. Since the GOAL team submitted the Brazilian timetabling instances they are not considered for ranking for the first four instances. Table 5.2 provides, also, the average ranks of each approach based on their ranking for each instance per run. The proposed hyper-heuristic turns out to be the second best approach.

Table 5.3 summarises the feasibility/objective values obtained by the five competitors' solvers including the proposed HySST multi-stage hyper-heuristic approach on the hidden instances on Round three. Considering the best solution encountered over a large number of runs with a variety of runtimes and parameter settings, the proposed multi-stage hyper-heuristic produced the best results in six including three ties out of eighteen instances. The GOAL team was not considered for the Brazilian instances for ranking in this round as well. Table 5.3 provides, also, the average ranks of each competing approach in round 3. Our selection hyper-heuristic became the second best approach.

5.2 A Hybrid Approach Embedding a Multi-stage Hyper-heuristic

By the time of the MISTA 2013, the Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance (DRW) multi-stage hyper-heuristic is proposed and used in a hybrid approach, that won the MISTA 2013 challenge at which the purpose is to solve the multi-mode resource-constrained multi-project scheduling problem

TABLE 5.2: The performance comparison of the HySST approach to the other competing approaches over 10 trials showing the best quality (cost) of a solution indicated as feasibility-value.objective-value in Round 2. The best values are highlighted in bold

Problem	KHE	HySST	GOAL	HFT	Lectio
Instance2	3.20001	1.00069	1.00051	5.00183	**0.00019**
Instance3	3.50002	0.00096	**0.00087**	26.00264	0.00112
Instance4 - Brazil	39.10001	2.00238	16.00104	63.00225	**1.00172**
Instance6	11.60003	2.00229	4.00207	21.00423	**0.00183**
ElementarySchool	9.90000	0.00004	**0.00003**	29.00080	**0.00003**
SecondarySchool2	1.80017	0.00006	**0.00000**	28.01844	0.00014
Aigio 1st HS 2010	12.20008	0.00322	**0.00006**	45.03665	0.00653
Instance4 - Italy	32.60218	0.04012	**0.00169**	250.05966	0.00225
Instance1	1307.10005	1065.17431	**38.09789**	986.42437	274.04939
Kottenpark2003	4.40747	**0.47560**	0.87084	203.87920	34.55960
Kottenpark2005A	32.70292	**26.35251**	27.37026	393.40463	185.83973
Kottenpark2008	72.51725	32.71562	**10.33034**	INVALID	84.99999
Kottenpark2009	48.22637	33.99999	**25.14030**	337.99999	97.96060
Woodlands2009	13.90000	2.00047	2.00012	59.00336	**0.00094**
School	2.50039	0.01247	**0.00597**	63.13873	0.01927
WesternGreeceUni3	0.00024	0.00010	**0.00005**	14.00198	30.00002
WesternGreeceUni4	0.00044	0.00016	**0.00005**	233.00277	35.00070
WesternGreeceUni5	15.40000	0.00001	**0.00000**	9.00174	4.00013
Average ranking		2.23	**1.18**	3.64	2.32

(MRCMPSP). The approach consists of a *construction phase* followed by an *improvement phase* using a memetic meta-heuristic method and a multi-stage hyper-heuristic. The memetic algorithm is used to generate a pool of eight solutions, in which a solution is improved through successive application of multi-stage hyper-heuristic algorithm.

Some preliminary experimental results are provided in Table 5.4. The entries in the *Comp.* columns are taken from the competition website for reference purposes. They demonstrate the best objective values achieved during the qualification (A instances) and final (B and X instances) phases of the competition. Furthermore, the *Avg.* column gives the mean performance of our approach over 2500 runs per instance under conditions similar to the competition. For purposes of comparison, the *Best* column provides the best values that we ever encountered during all our experiments. The experiments are performed on a group of identical 64-bit Intel i7 (3.2 GHz) machines where each machine has 16 GB of RAM operating on Microsoft Windows operating system.

It is worth mentioning that during the final phase where the competing algorithms were applied on B and X instance classes only, our approach found the best solution for 17

TABLE 5.3: The best-of-runs performance comparison of the HySST approach to the other competing approaches using the quality (cost) of a solution indicated as feasibility-value.objective-value in Round 3. The best values are highlighted in bold

Dataset	HySST	GOAL	HFT	Lectio	VAGOS
Instance2	0.00044	0.00032	0.00082	**0.00005**	0.00026
Instance3	0.00084	0.00101	0.00212	0.00048	**0.00047**
Instance4 - Brazil	0.00176	1.00136	0.00205	0.00090	**0.00078**
Instance6	0.00150	0.00160	0.00347	**0.00060**	0.00074
ElementarySchool	**0.00003**	**0.00003**	**0.00003**	**0.00003**	ABSENT
SecondarySchool2	**0.00000**	**0.00000**	0.00576	**0.00000**	ABSENT
Aigio 1st HS 2010	0.00218	**0.00000**	0.00555	0.00076	ABSENT
Instance4 - Italy	**0.00052**	0.00061	0.08623	0.00078	ABSENT
Instance1	0.01721	**0.00003**	36.12987	274.00281	ABSENT
Kottenpark2003	0.03919	0.05355	1.88983	**0.02918**	ABSENT
Kottenpark2005A	**15.28693**	24.13930	36.36132	198.04845	ABSENT
Kottenpark2008	16.17720	**10.27909**	167.99999	129.69216	ABSENT
Kottenpark2009	**18.08010**	19.05590	148.99999	87.09440	ABSENT
Woodlands2009	0.00013	**0.00012**	8.00206	0.00019	ABSENT
School	0.00920	**0.00441**	1.08163	0.00762	ABSENT
WesternGreeceUni3	0.00007	**0.00005**	0.00032	30.00002	**0.00005**
WesternGreeceUni4	0.00009	**0.00008**	0.00142	35.00058	ABSENT
WesternGreeceUni5	**0.00000**	**0.00000**	0.00064	4.00001	**0.00000**
Average ranking	2.25	**1.64**	3.75	2.75	3.86

out of 20 instances and performs the best in around 90% of all the trials. That is, except for instances B-2, B-4 and X-1, all the *Comp.* entries in Table 5.4 are achieved by our approach. Furthermore, it is clear from Table 5.4 (*Best* entries) that our algorithm achieved the best solution for all instances during 2500 run per instance experiments. The proposed algorithm significantly outperformed the other approaches during the MISTA 2013 challenge with a mean rank of 1.1 for multi-mode resource-constrained multi-project scheduling.

To illustrate the performance of our approach at various stages of the search, two box-plots are provided in Figures 5.4 and 5.5 for B-1 and X-10 instances respectively. These instances have been chosen to demonstrate the performance of our algorithm on relatively small and large instances. However, our experiments show that the algorithm behaves in a similar manner with respect to other instances in such that, the method rapidly improves the quality of the solution in hand; after a while, the improvement process slows down. The central dividing line of each box in each of the figures, presents the median objective value obtained by our algorithm. The edges of each box refer

TABLE 5.4: Summary of experimental results on MISTA 2013 instances. The objective values are given as ordered pairs of "TPD", total project delay, and "TMS", total makespan. The 'Comp.' values are from the competition website for all the entrants in the qualification and final rounds. 'Avg.' values are average values for our algorithm under time and machine conditions intended to match those of the competition. 'Best' is the best solution encountered over a large number of runs

	TPD			TMS		
Instance	Avg.	Comp.	Best	Avg.	Comp.	Best
A-1	1	1	1	23	23	23
A-2	2	2	2	41	41	41
A-3	0	0	0	50	50	50
A-4	65	65	65	42	42	42
A-5	155	153	150	105	105	103
A-6	141	147	133	808	96	99
A-7	605	596	590	201	196	190
A-8	292	302	272	153	155	148
A-9	208	223	197	128	119	122
A-10	880	969	836	313	314	303
B-1	352	349	345	128	127	124
B-2	452	434	431	167	160	158
B-3	554	545	526	210	210	200
B-4	1299	1274	1252	283	289	275
B-5	832	820	807	255	254	245
B-6	950	912	905	232	227	225
B-7	802	792	782	232	228	225
B-8	3323	3176	3048	545	533	523
B-9	4247	4192	4062	754	746	738
B-10	3290	3249	3140	455	456	436
X-1	405	392	386	143	142	137
X-2	356	349	345	164	163	158
X-3	329	324	310	193	192	187
X-4	960	955	907	209	213	201
X-5	1785	1768	1727	373	374	362
X-6	730	719	690	238	232	226
X-7	866	861	831	233	237	220
X-8	1256	1233	1201	288	283	279
X-9	3272	3268	3155	648	643	632
X-10	1613	1600	1573	383	381	373

to 25^{th} and 75^{th} percentiles while the whiskers (demonstrated by a + marker) are the extreme objective values which are not considered as outliers. Also, the curve which passes through the plot demonstrates the average performance of the algorithm.

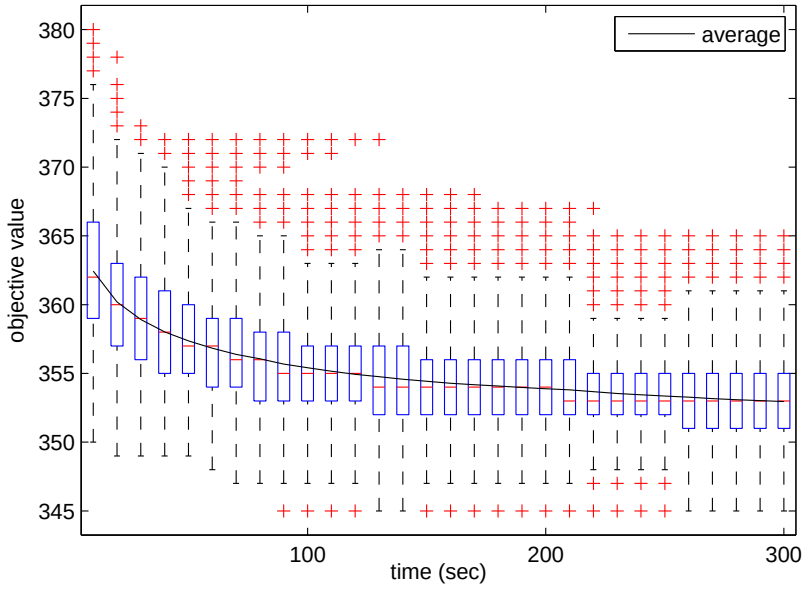


FIGURE 5.4: Performance of the proposed approach on instance B-1

5.2.1 Comparison of Different Approaches

The proposed hybrid approach is evaluated in this section. For convenience, the winning hybrid approach is denoted as MCTS-DRW. This approach is used as a baseline to validate the effectiveness and performance of other approaches.

The Monte-Carlo Tree Search (MCTS) constructor method can be replaced by random constructor (RC) method and the multi-stage hyper-heuristic (DRW) method can be replaced by a local search/hill climbing (LS) method which chooses a random heuristic at each step, applies it to the current solution and accepts if the new solution makes a strict improvement. Additionally, the roulette wheel selection hyper-heuristic (S1HH) can be employed alone without enabling the second dominance-based hyper-heuristic (S2HH) turning the multi-stage hyper-heuristic into a simple random hyper-heuristic. The developed MCTS-DRW approach is compared against a combination of five different approaches: (1) random constructor and local search (RC-LS), (2) random constructor

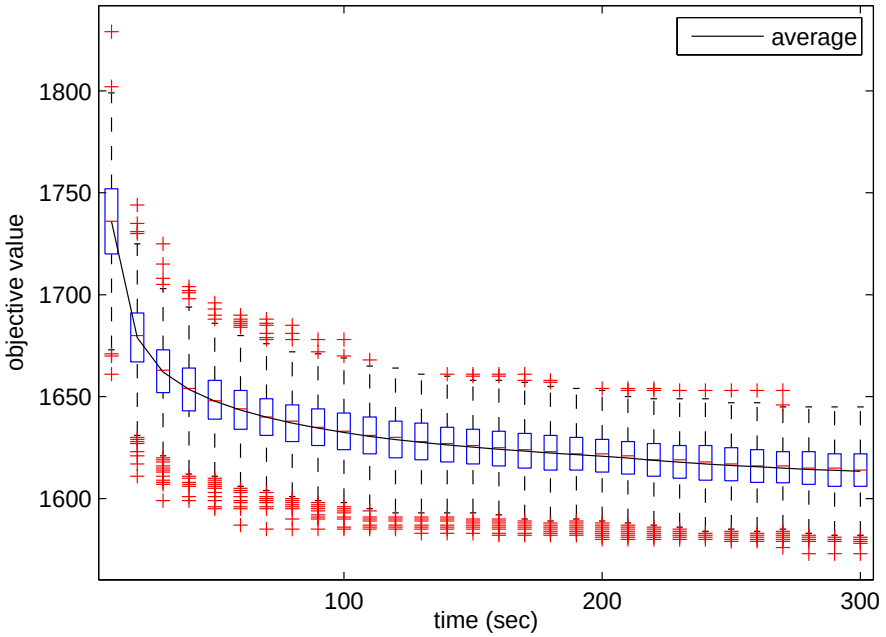


FIGURE 5.5: Performance of the proposed approach on instance X-10

and simple random hyper-heuristic (RC-S1HH), (3) random constructor and multi-stage hyper-heuristic (RC-DRW), (4) MCTS constructor and local search (MCTS-LS) and (5) MCTS constructor and simple random hyper-heuristic (MCTS-S1HH). For each instance, 10 runs are executed each for 300 seconds, and the average objective, standard deviation and the best objective values are calculated. All algorithms are ranked based on the objective values obtained at the end of each run. To compare the pairwise performance between any given two approaches (A and B) statistically, we used the Mann-Whitney-Wilcoxon test [181]. Four notations are considered: $>$, $<$, $\geq$ and $\leq$. A $>$ ($<$) B indicates that A (B) is statistically better than B (A) and this performance difference is statistically significant within a confidence interval of 95% and A $\geq$ ($\leq$) B denotes that A (B) is slightly better than B (A) on average.

For the fairness, all the approaches are written in C++ and performed on Intel(R) Core(TM) i7-3930K with a 3.20 GHz and 16.00GB of RAM. For the first four instances of set A, all the approaches deliver solutions with the same quality. Such instances are not reported in the results presented. Table 5.5 summarises the results.

TABLE 5.5: The average ranking and the performance comparison of MCTS-DRW, RC-LS, RC-S1HH, RC-DRW, MCTS-LS and MCTS-S1HH methods based on the average (avg.), standard deviation (std.), minimum (min.) of the objective values over 10 runs and the pairwise average performance comparison of MCTS-DRW vs other approaches based on Mann-Whitney-Wilcoxon for each instance produced by each approach. The best avr. and min. values per each instance are highlighted in bold

	MCTS-DRW				RC-LS				RC-S1HH				RC-DRW				MCTS-LS				MCTS-S1HH		
Instance	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.
A5	152.505	0.516	152.104	>	163.910	2.700	160.108	≤	152.505	0.843	151.105	≤	**152.304**	1.033	**150.104**	>	164.209	3.696	161.108	≥	152.605	0.850	151.105
A6	140.075	2.541	136.92	>	165.103	5.313	155.102	≤	**138.337**	1.796	**134.91**	≤	138.842	2.136	**134.91**	>	164.391	4.018	157.100	≥	141.454	2.474	137.940
A7	603.700	4.143	597.201	>	642.407	4.803	633.206	≤	**603.501**	2.791	598.203	≥	604.400	3.329	**597.194**	>	633.808	9.252	618.208	≥	605.001	2.618	602.197
A8	289.954	2.573	286.154	>	341.163	6.733	330.163	≤	289.453	3.684	285.152	≤	**289.353**	3.766	**280.152**	>	343.762	11.039	328.158	≥	291.953	5.287	283.152
A9	206.928	3.553	**201.126**	>	241.135	5.354	232.133	≥	207.928	1.751	204.127	≤	**205.629**	2.173	202.127	>	241.535	5.817	229.132	≤	206.927	3.011	201.128
A10	880.912	5.168	874.312	>	984.631	20.539	946.328	≤	877.114	10.802	859.311	≥	881.513	7.886	866.310	>	984.630	14.982	956.334	≤	**877.112**	10.718	**857.311**
B1	351.328	1.550	348.126	>	367.533	4.970	360.134	≥	351.729	2.633	349.132	≥	352.028	3.178	**345.126**	>	364.533	3.240	358.128	≤	**350.928**	1.549	348.128
B2	**448.367**	6.957	**439.167**	>	521.083	11.949	504.180	≥	449.067	7.064	440.164	≥	452.567	5.378	444.167	>	512.980	12.589	501.179	≥	449.567	5.541	443.167
B3	552.011	4.158	545.210	>	623.820	6.186	612.219	≤	**548.809**	4.223	542.209	≤	551.909	5.736	544.213	>	620.920	12.921	594.216	≤	549.710	6.382	**541.207**
B4	1302.582	9.946	1287.282	>	1357.493	9.908	1342.299	≤	1297.183	9.667	1279.279	≤	1293.884	15.358	**1270.287**	>	1354.594	11.245	1338.292	≤	**1293.881**	11.207	1283.281
B5	**828.054**	3.675	823.254	>	903.164	9.364	888.260	≥	830.253	7.861	**816.252**	≥	830.457	9.053	818.257	>	894.866	7.548	885.268	>	833.255	5.207	824.257
B6	**945.031**	12.684	927.230	>	1094.848	27.677	1042.242	≥	948.530	13.930	**921.226**	≥	952.632	10.638	935.232	>	1097.249	19.974	1073.244	≥	958.832	15.774	929.230
B7	800.232	5.639	**792.231**	>	858.239	16.169	842.237	≤	**800.132**	4.434	796.232	≥	803.432	4.468	796.230	>	863.439	17.127	834.236	≥	804.532	3.684	798.232
B8	3324.245	36.669	3287.540	>	3618.982	38.844	3564.579	≤	3319.546	44.914	3265.549	≤	**3303.446**	92.207	**3119.538**	>	3638.986	60.855	3558.580	≥	3324.246	79.502	3184.540
B9	4259.457	45.052	4184.751	>	4593.998	45.657	4523.786	≤	**4222.853**	48.660	**4128.744**	≤	4245.954	43.679	4151.747	>	4629.701	42.432	4572.787	≤	4242.155	41.102	4181.759
B10	3273.255	47.137	**3175.449**	>	3481.880	97.797	3255.463	≥	3302.457	39.823	3222.446	≥	3275.154	43.377	3215.445	>	3508.582	57.163	3437.473	≤	**3251.553**	34.272	3208.450
X1	**403.042**	7.638	**393.141**	>	433.949	7.843	417.146	≥	403.143	5.638	395.142	≥	404.742	4.880	397.144	>	433.350	8.123	419.148	≥	403.943	4.341	399.143
X2	353.165	4.498	346.163	>	396.872	6.291	386.168	≥	353.764	3.308	347.161	≤	**351.763**	3.565	**344.163**	>	399.772	6.347	390.169	≥	355.965	2.251	352.163
X3	325.994	4.591	317.193	>	386.202	7.747	376.199	≥	327.893	4.644	319.192	≤	**325.592**	5.680	317.194	>	390.003	6.252	379.202	≥	326.093	4.203	**317.189**
X4	957.609	9.935	939.212	>	1021.717	12.060	1005.211	≥	958.610	12.096	930.205	≤	**950.309**	16.698	**922.205**	>	1017.317	13.666	997.219	≤	955.609	14.857	937.205
X5	1784.372	18.270	**1759.375**	>	1893.788	22.360	1861.390	≥	1785.374	17.702	1762.373	≤	**1783.573**	16.144	1761.375	>	1902.591	19.187	1868.396	≥	1786.473	12.957	1769.368
X6	731.037	10.053	713.237	>	846.055	19.491	813.250	≥	733.438	9.055	719.238	≥	731.936	8.593	717.228	>	845.057	17.940	821.255	≤	**727.439**	12.588	**702.233**
X7	862.731	10.244	**840.235**	>	909.543	10.067	891.241	≤	862.133	6.083	850.226	≤	**861.434**	6.927	851.235	>	909.741	6.620	899.238	≥	864.032	5.791	854.236
X8	1252.788	12.617	1233.290	>	1378.705	21.234	1357.296	≥	1255.088	11.459	1242.286	≤	1248.686	15.360	**1223.278**	>	1383.504	14.353	1358.303	≤	**1247.088**	9.705	1232.288
X9	3266.747	39.002	3217.642	>	3564.585	29.789	3524.676	≥	3294.950	20.848	3257.638	≤	**3253.047**	37.976	**3186.638**	>	3576.585	62.744	3465.671	≤	3262.648	37.394	3213.641
X10	**1609.383**	6.219	1599.386	>	1712.603	15.770	1687.401	≥	1616.282	11.997	1599.375	≥	1610.684	12.105	**1592.385**	>	1716.304	17.183	1679.402	≥	1613.584	10.903	1600.380
rank	2.52				5.48				2.48				2.45				5.51				2.56		

Examining Table 5.5, it can be observed from the computational results based on the different experimental scenarios that all of the problem instances solved using the local search as a component are worse than those solved by the other approaches and this performance is statistically significant. Concerning the remaining approaches, the performance variations are not statistically significant within a confidence interval of 95% based on the Mann-Whitney-Wilcoxon test. However, based on the competition ranking method, RC-DRW seems to perform the best. Considering both MCTS and RC constructor methods, DRW performs better than S1HH on 15 (out of 26) instances on average. Therefore, the incorporation of the dominance-based strategy in the multi-stage hyper-heuristic appears to play a role of solving the problem in relatively effective manner in some instances.

5.2.2 Performance Analysis of MCTS-DRW and RC-DRW

In this section, we compare the MCTS-DRW and RC-DRW algorithms on B2 and X1 (in which MCTS-DRW is performing good on average) and on B8 and X4 (in which RC-DRW is performing good on average) instances. Table 5.6 summarises the results. In all the cases, the time it takes to construct initial solutions using RC is much faster than using MCTS; but the quality of the generated solutions is much worse than the solutions generated by MCTS. By the end of the search, RC-DRW seems to deliver better quality of solutions than MCTS-DRW on most of the cases.

TABLE 5.6: The performance comparison of the constructor methods (MCTS and RC) based on the 'time' they take to construct a population of solutions in second, the average objective values of the initially generated solutions '$O_{initial}$', and the best objective values obtained after applying the improvement phase 'O_{best}'

	MCTS-DRW			RC-DRW		
Instance	time	$O_{initial}$	O_{best}	time	$O_{initial}$	O_{best}
B2	0.748	1167.5	445.167	0.000	1763.375	446.165
B8	1.638	9294.875	3358.533	0.015	14221.625	3220.531
X1	0.265	1284.625	410.141	0.000	1645	397.143
X4	0.390	2556.25	979.211	0.000	3746.875	968.218

The behaviour of the DRW multi-stage hyper-heuristic on 8 solutions of the population on the selected instances is illustrated in Figures 5.6-5.9. We suspect that using the MCTS constructor method makes the multi-stage hyper-heuristic somehow to give

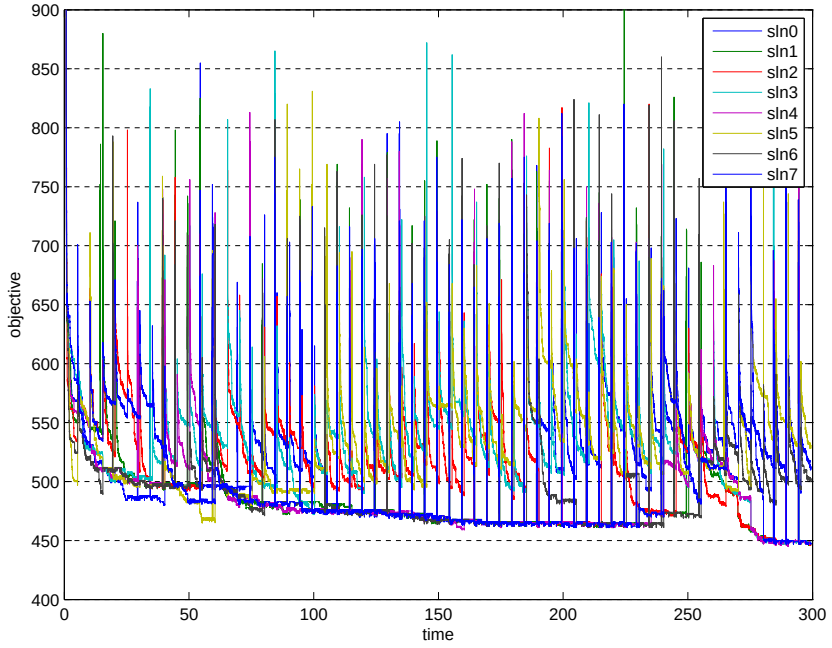
locally optimal solutions faster than in the RC constructor. After this point, the improvements to the best obtained solutions slow down. Refreshing the population in the memetic meta-heuristic method and the fluctuations of the objective values due to the employment of the threshold move acceptance in the multi-stage hyper-heuristic seem to help the search process to slightly improve the quality of the solutions after this point. The figures provide evidence that the RC generates solutions that assists to avoid the possibility of getting stuck early, and in general, it delivers better quality of solutions than in MCTS. This conclusion cannot be generalised to all instances where at some instances, MCTS performs slightly better.

The percentage utilisation with respect to the number of times a low level heuristic gets selected considering only improving moves is shown in Figures 5.10-5.13. Almost the same phenomenon is observed in all the selected instances considering the two approaches. The large set of low level heuristics (LLH13-16) does not seem to provide consistent improvements to the quality of solutions during the search process. Most of the improvements are due to LLH1 (InsertJob), LLH2 (SetMode) and the hill climber LLH12 (FILS setMode), across all instances. The remaining low level heuristics contribute (almost) equally in improving the candidate solutions.

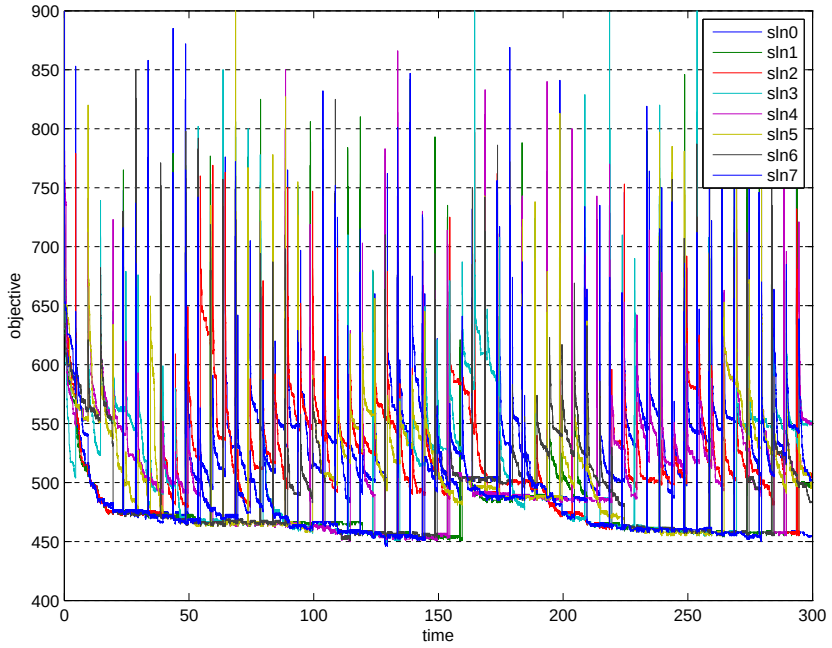
5.3 Testing the Level of Generality of Multi-stage Hyper-heuristics on HyFlex Problems

We have evaluated the performance of the proposed Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance, denoted as MSHH, across six problem domains of HyFlex. During our experimentation, crossover operators are ignored as low level heuristics, considering that the multi-stage hyper-heuristics operate under a single point based search framework. The Mann-Whitney-Wilcoxon test [181, 182] is used as a statistical test for pairwise average performance of two given algorithms. We have used the following notation: Given two algorithms; A versus B, $>$ ($<$) denotes that A (B) is better than B (A) and this performance difference is statistically significant within a confidence interval of 95% and A $\geq$ B (A $\leq$ B) indicates that A (B) performs slightly better than B (A).

Thirty one runs are repeated using each algorithm unless it is mentioned otherwise. A benchmarking software tool provided at the CHeSC 2011 website is used to obtain the

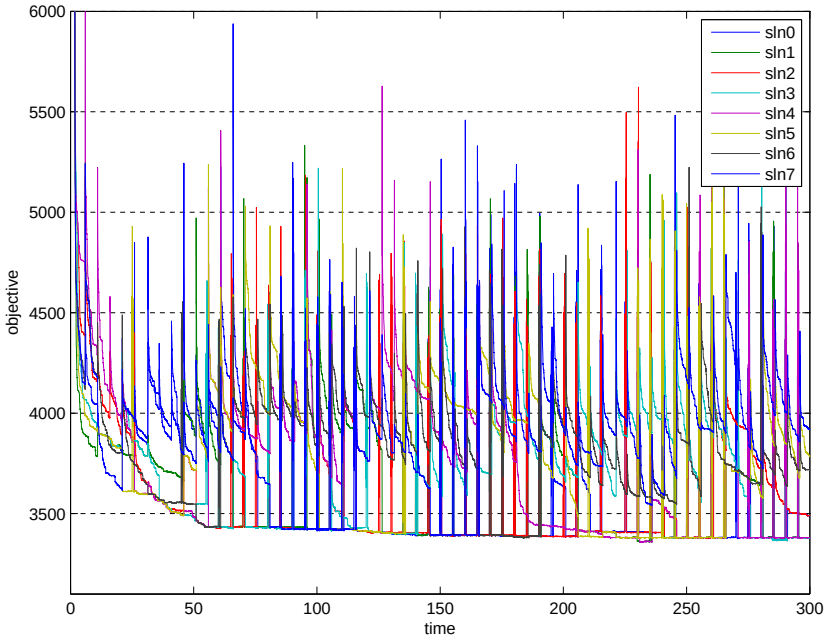


(a) MCTS-DRW

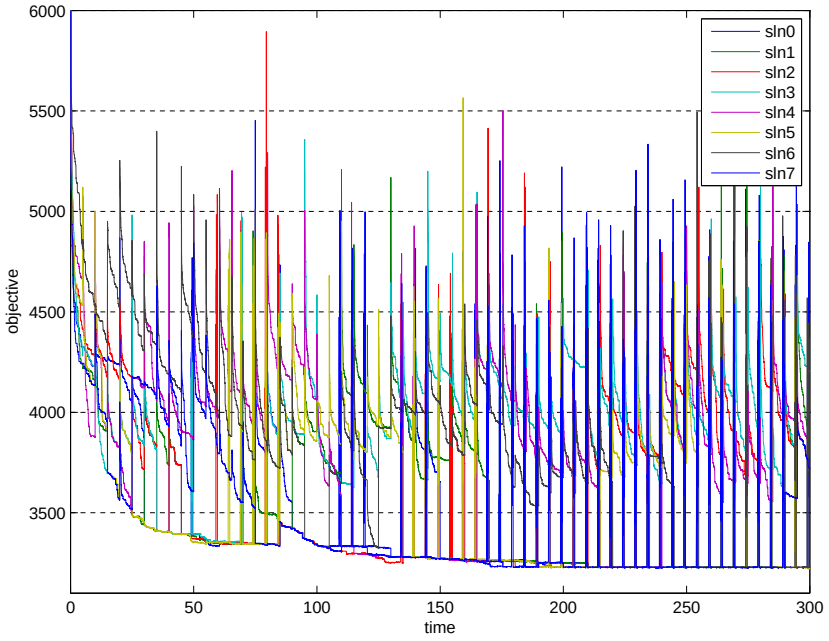


(b) RC-DRW

FIGURE 5.6: Plots of the objective values of the solutions in the population versus time while solving instance B2

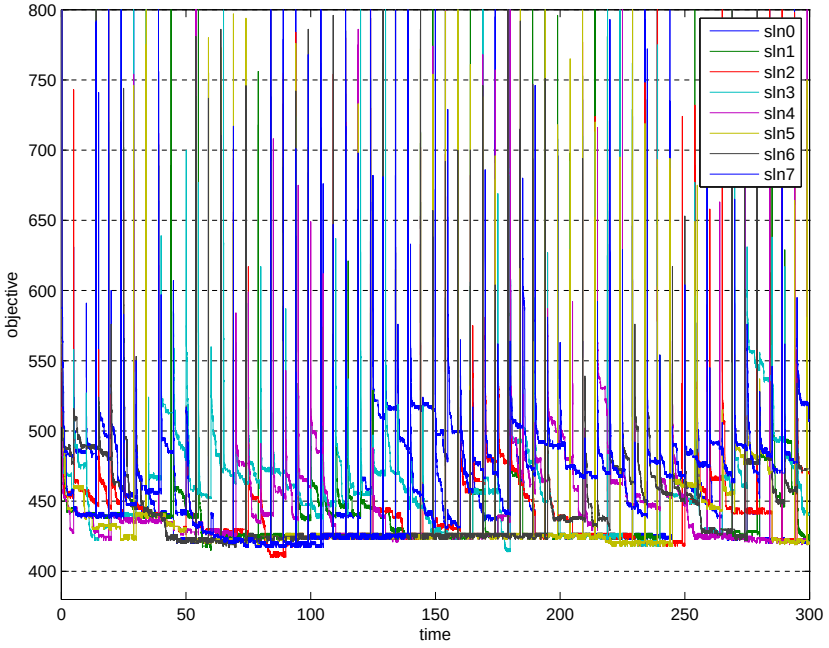


(a) MCTS-DRW

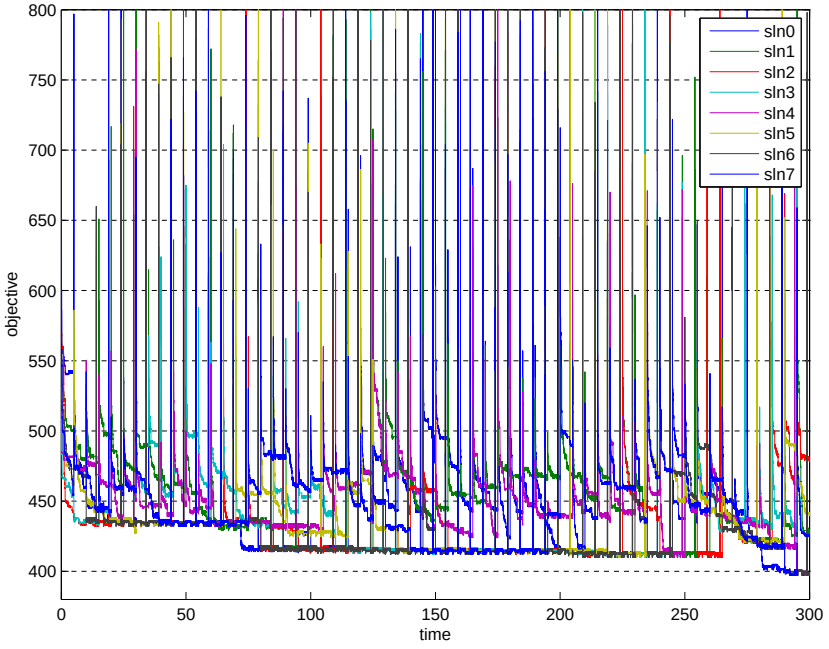


(b) RC-DRW

FIGURE 5.7: Plots of the objective values of the solutions in the population versus time while solving instance B8

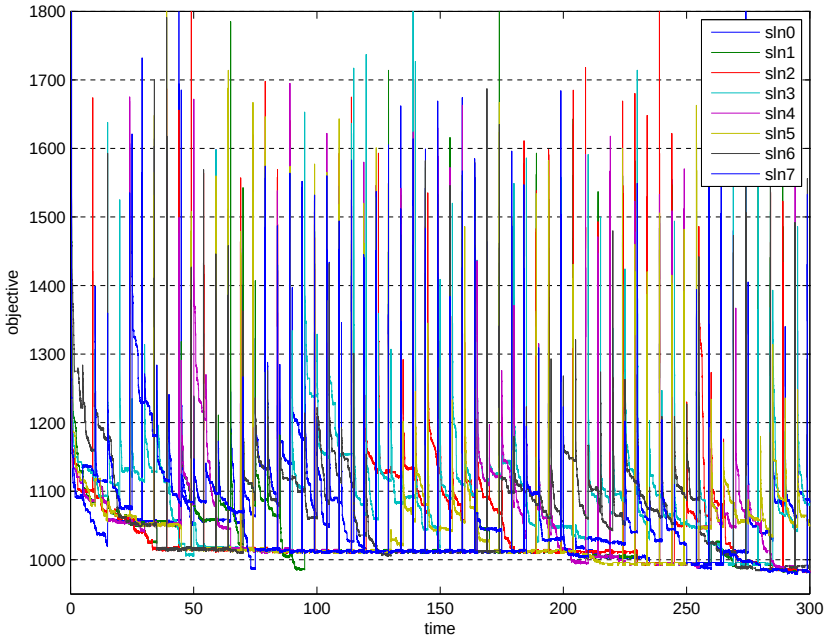


(a) MCTS-DRW

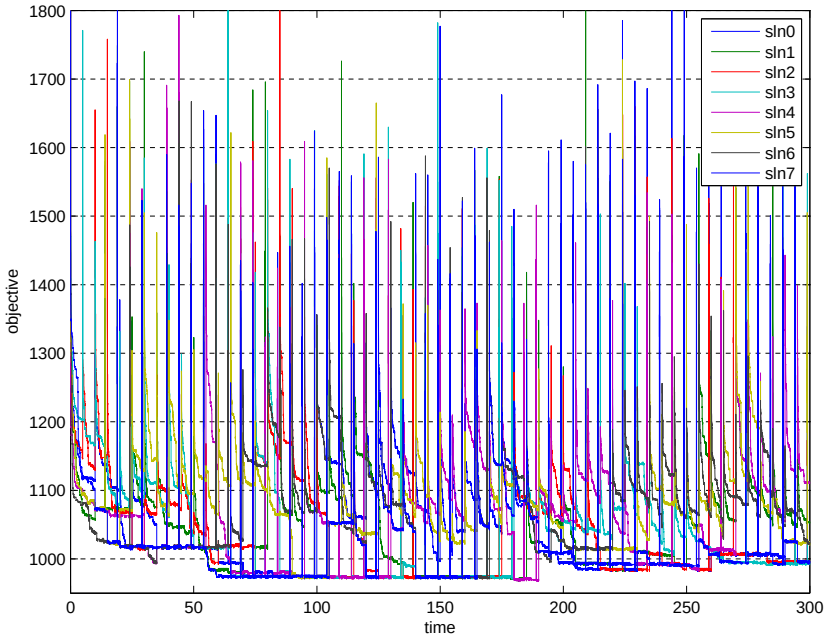


(b) RC-DRW

FIGURE 5.8: Plots of the objective values of the solutions in the population versus time while solving instance X1

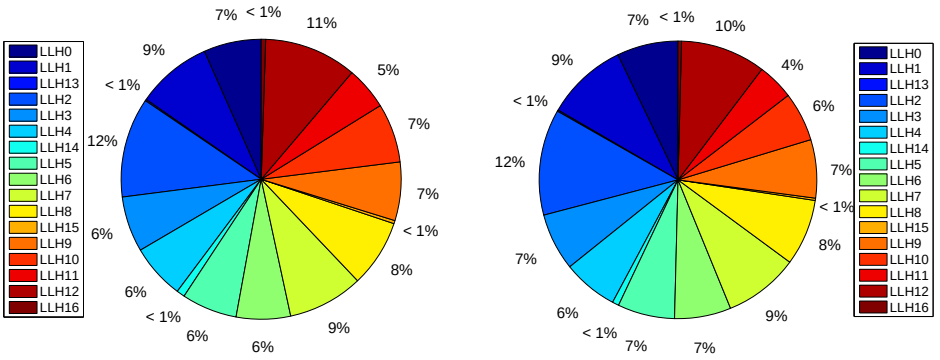


(a) MCTS-DRW



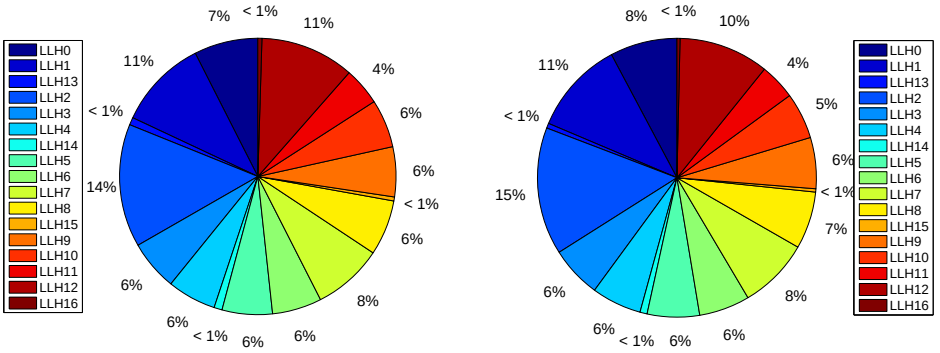
(b) RC-DRW

FIGURE 5.9: Plots of the objective values of the solutions in the population versus time while solving instance X4



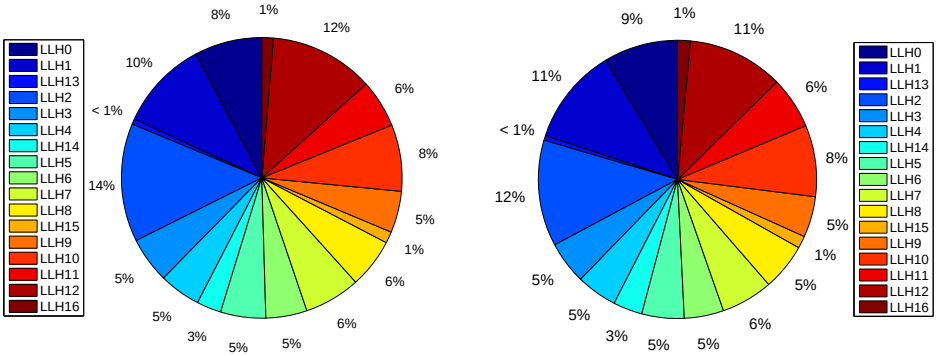
(a) MCTS-DRW (b) RC-DRW

FIGURE 5.10: Average percentage utilisation of the low level heuristics while solving instance B2



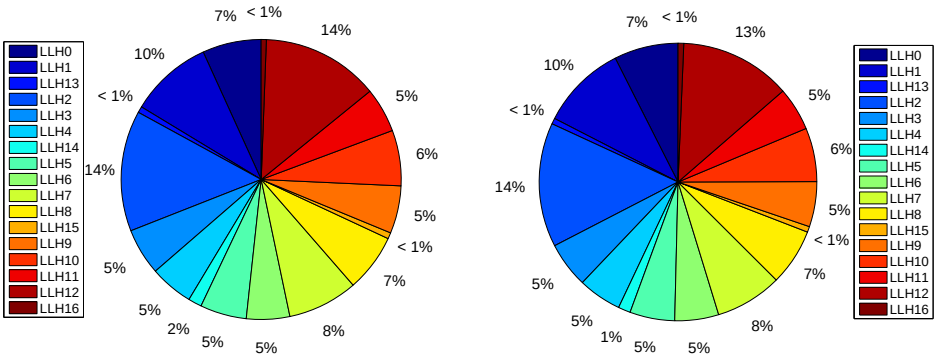
(a) MCTS-DRW (b) RC-DRW

FIGURE 5.11: Average percentage utilisation of the low level heuristics while solving instance B8



(a) MCTS-DRW (b) RC-DRW

FIGURE 5.12: Average percentage utilisation of the low level heuristics while solving instance X1



(a) MCTS-DRW (b) RC-DRW

FIGURE 5.13: Average percentage utilisation of the low level heuristics while solving instance X4

equivalent time value (*timeLimit*) in the used machines that correspond to 600 seconds according to the competition rule.

We have fixed the parameter values based on intuition from our previous works [43, 170, 171, 173]: $\tau = 15ms$, $d = 9s$, $s_1 = 20s$, $s_2 = 5$, $P_{S2HH} = 0.3$, $C = \{0, 3, 6, 9\}$. The experiments are performed using those settings as "regular" settings on all thirty instances from all domains used at CHeSC 2011. We compare the performance of our approach to each individual hyper-heuristic used in a stage, the other proposed multi-stage hyper-heuristics and competing hyper-heuristics of CHeSC 2011 including the state-of-the art hyper-heuristic (denoted as *AdapHH*) which won the competition, respectively.

5.3.1 Parameter Settings

A set of experiments is performed on four arbitrarily chosen (first) instances of four public problem domains to observe the performance of the proposed algorithm under different parameter settings:

- $\tau = \{10, \mathbf{15}, 20, 30\}$ (in milliseconds)
- $d = \{7, \mathbf{9}, 10, 12\}$ (in seconds)
- $s_1 = \{10, 15, \mathbf{20}, 25\}$ (in seconds)
- $s_2 = \{3, \mathbf{5}, 10, 15\}$ (in steps/iterations)
- $P_{S2HH} = \{0.1, \mathbf{0.3}, 0.6, 0.9, 1.0\}$

- $C = \{\{0\}, \{3\}, \{6\}, \{9\}, \{\mathbf{0, 3, 6, 9}\}\}$

While testing a different setting for a given parameter, the remaining parameters are fixed with the values marked in bold which are our initial settings. MSHH is run with each setting for 10 trials on the selected instance from each public domain. Table 5.7 summarises the results based on the average performance of MSHH with different parameter settings. In all cases, MSHH with the "regular" parameter setting wins against another setting, however, mostly, this performance difference is not statistically significant. There are a few cases in which MSHH with a setting other than the proposed one yields a slightly better average performance on the BP and PS instances. For example, $\tau = 10$ performs slightly better than $\tau = 15$ on the BP instance, and $P_{S2HH} = 0.6$ is a slightly better choice than $P_{S2HH} = 0.3$ for the PS instance. MSHH with the "regular" parameter setting always performs better than another setting on the PFS and SAT instances. In the overall, MSHH with the "regular" parameter setting based on intuition turns out to be indeed a good choice and so the same settings are used in the remaining experiments.

TABLE 5.7: The average performance comparison of MSHH for different parameter settings over 10 trials. MSHH with "regular" setting of a given parameter is compared to MSHH when that setting is changed to a given setting based on Mann-Whitney-Wilcoxon statistical test for each selected instance from a public domain

Par.:	τ			d			C			
Dom.	10	20	30	7	10	12	{0}	{3}	{6}	{9}
SAT	$\geq$	$\geq$	$\geq$	$>$	$\geq$	$\geq$	$>$	$>$	$\geq$	$\geq$
BP	$\leq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\leq$	$\geq$	$\geq$
PS	$\geq$	$\geq$	$>$	$\leq$	$\geq$	$\geq$	$\leq$	$\geq$	$\geq$	$\geq$
PFS	$>$	$>$	$>$	$\geq$	$>$	$\geq$	$>$	$>$	$\geq$	$\geq$
wins	3	4	4	3	4	4	3	3	4	4

Par.:	s_1			s_2			P_{S2HH}			
Dom.	10	15	25	3	10	15	0.1	0.6	0.9	1
SAT	$\geq$	$\geq$	$\geq$	$\geq$	$>$	$\geq$	$>$	$\geq$	$\geq$	$\geq$
BP	$\leq$	$>$	$\leq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\leq$	$\geq$
PS	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$>$	$\leq$	$>$	$\geq$
PFS	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$\geq$	$\geq$
wins	3	4	3	4	4	4	4	3	3	4

5.3.2 Performance Comparison to the Constituent Hyper-heuristics

We have experimented with the hyper-heuristics used at each stage, denoted as S1HH and S2HH, respectively, run on their own and compare their performances to the performance of the proposed multi-stage hyper-heuristic. Tables 5.8 presents the results. MSHH obtains the best solution in 31 trials for 27 out of 30 of the CHeSC 2011 instances, which include all instances from the SAT, BP and TSP domains and exclude one instance from the remaining domains. On average, MSHH still performs better than the constituent hyper-heuristics of S1HH and S2HH run on their own on the 22 instances across all six problem domains. The standard deviation associated with the average objective function value of MSHH is the lowest in all cases on the SAT and TSP problem domains.

On average, MSHH outperforms S2HH and this performance is statistically significant for all instances, except for Inst2, Inst3 and Inst4 from the PS domain and Inst3 from the VRP domain. On the SAT and TSP domains, MSHH performs still significantly better than S1HH on all instances. On PS, MSHH is better than S1HH in four instances, but this performance variation is significant for two out of the four instances. MSHH performs slightly better than S1HH on the BP, PFS and VRP domains in the overall. However, S1HH performs better than MSHH only on two instances, Inst1 from BP and Inst2 from PFS for which the performance difference is statistically significant.

Our study empirically confirms that combining hyper-heuristics under a multi-stage framework can potentially lead to an improved overall performance.

5.3.3 Performance Comparison to Multi-stage Hyper-heuristics

The performance of the proposed Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) is compared to the performance of the proposed elaborate and successful multi-stage hyper-heuristics which are described in Chapter 4: Greedy-gradient - Simulated Annealing Hyper-heuristic (also known as greedy-gradient) (GGHH) [28], Dominance-based Random Descent/Gradient Hyper-heuristic with Naïve Move Acceptance (DRD) [170], Robinhood Hyper-heuristic with an Adaptive Threshold Acceptance (RHH) [171], Selection Hyper-heuristic with an Adaptive Threshold Acceptance (HySST) [43] and Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance (DRW)

TABLE 5.8: The performance comparison of MSHH, S1HH and S2HH based on the average (avg.), associated standard deviation (std.), minimum (min.) of the objective values over 31 trials and the pairwise average performance comparison of MSHH vs S1HH and MSHH vs S2HH based on Mann-Whitney-Wilcoxon for each CHeSC 2011 instance produced by each approach. The hyper-heuristic producing the best value for avr. and min. per each instance are highlighted in bold

		MSHH					S1HH				S2HH		
Domain	Instance	avg.	std.	median	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.
SAT	Inst1	**0.9**	0.7	1.0	**0.0**	$>$	6.4	4.5	1.0	$>$	15.0	4.6	3.0
	Inst2	**3.1**	3.9	2.0	**1.0**	$>$	21.3	13.3	3.0	$>$	44.9	9.8	18.0
	Inst3	**0.7**	0.5	1.0	**0.0**	$>$	7.1	7.7	**0.0**	$>$	26.3	14.0	1.0
	Inst4	**1.7**	1.0	1.0	**1.0**	$>$	5.7	4.3	**1.0**	$>$	20.0	4.6	12.0
	Inst5	**7.6**	0.9	7.0	**7.0**	$>$	10.4	1.5	**7.0**	$>$	15.4	1.7	13.0
BP	Inst1	0.0163	0.0014	0.0163	**0.0136**	$<$	**0.0159**	0.0010	0.0137	$>$	0.0198	0.0015	0.0160
	Inst2	**0.0037**	0.0015	0.0030	**0.0025**	$>$	0.0061	0.0015	0.0034	$>$	0.0104	0.0021	0.0077
	Inst3	**0.0050**	0.0015	0.0049	**0.0025**	$\geq$	0.0054	0.0012	0.0027	$>$	0.0128	0.0011	0.0104
	Inst4	0.1084	0.0000	0.1084	**0.1083**	$\leq$	**0.1084**	0.0000	0.1083	$>$	0.1084	0.0000	0.1084
	Inst5	**0.0050**	0.0019	0.0044	**0.0032**	$\geq$	0.0055	0.0021	0.0032	$>$	0.0210	0.0015	0.0187
PS	Inst1	**25.5**	4.5	25.0	**16.0**	$>$	28.8	4.7	18.0	$>$	31.6	4.9	22.0
	Inst2	9668.9	217.8	9638.0	**9184.0**	$\leq$	**9645.3**	159.6	9334.0	$\leq$	9645.8	106.7	9391.0
	Inst3	**3283.7**	93.3	3270.0	**3132.0**	$\geq$	3304.8	99.6	3134.0	$\geq$	3309.9	110.2	3172.0
	Inst4	**1786.3**	172.1	1760.0	1545.0	$\geq$	1801.0	142.3	1570.0	$\geq$	1836.0	291.1	**1400.0**
	Inst5	**353.2**	21.2	350.0	**315.0**	$>$	724.4	657.3	320.0	$>$	810.7	621.5	360.0
PFS	Inst1	**6239.8**	14.9	6239.0	**6212.0**	$>$	6287.6	21.9	6249.0	$>$	6353.3	29.8	6301.0
	Inst2	26895.2	55.3	26889.0	**26775.0**	$<$	**26873.2**	30.7	26822.0	$>$	26976.9	54.7	26849.0
	Inst3	**6333.8**	19.0	6325.0	**6303.0**	$>$	6360.5	16.4	6323.0	$>$	6405.5	23.7	6369.0
	Inst4	**11363.8**	32.7	11359.0	**11320.0**	$>$	11429.9	43.8	11357.0	$>$	11529.3	35.9	11436.0
	Inst5	26711.9	47.0	26709.0	26630.0	$\leq$	**26693.1**	40.7	**26608.0**	$>$	26779.1	49.8	26702.0
TSP	Inst1	**48208.1**	31.8	48194.9	**48194.9**	$>$	50032.0	571.1	49263.1	$>$	50326.5	606.6	49221.6
	Inst2	**2.09e^{+7}**	9.05e^{+4}	2.09e^{+7}	**2.07e^{+7}**	$>$	2.14e^{+7}	1.12e^{+5}	2.12e^{+7}	$>$	2.13e^{+7}	1.05e^{+5}	2.11e^{+7}
	Inst3	**6809.1**	7.1	6808.8	**6796.6**	$>$	7012.5	30.4	6964.6	$>$	7040.2	31.3	6988.6
	Inst4	**66840.2**	276.5	66843.6	**66236.8**	$>$	68908.4	382.4	68159.9	$>$	70241.9	704.6	68791.0
	Inst5	**53011.4**	469.7	52910.2	**52341.3**	$>$	54411.1	595.1	53686.0	$>$	55814.8	946.4	53992.4
VRP	Inst1	70998.4	3840.3	70506.5	**63948.2**	$\leq$	**70223.0**	2960.2	64273.2	$>$	84103.9	7225.8	68958.3
	Inst2	**13421.8**	251.6	13359.6	**13303.9**	$\geq$	13658.0	471.4	13319.6	$>$	13695.8	473.9	13320.0
	Inst3	148498.2	1625.8	148436.2	145466.5	$\leq$	**148232.6**	1935.3	145426.5	$\geq$	149553.2	2377.8	**145362.7**
	Inst4	21016.4	488.2	20671.4	**20650.8**	$\leq$	**20991.3**	478.0	20653.5	$>$	21131.9	510.3	20657.5
	Inst5	**148813.7**	1272.5	149193.7	**146334.6**	$\geq$	148999.1	1217.1	146844.9	$>$	150282.6	1616.3	146666.9

[173]. Table 5.9 presents the results achieved after the application of all those multi-stage hyper-heuristics to the CHeSC 2011 domains under the same setting. In the overall, MSHH turns out to be a viable general methodology outperforming the other multi-stage hyper-heuristic approaches in most of the HyFlex problem domains. The MSHH consistently performs the best in SAT, BP and TSP problem domains based on the average and minimum objective values obtained over 31 runs for each instance. Only for Inst1 from BP, DRD performs better in terms of average and minimum objective values. MSHH achieves the best average results on three instances on the PS and PFS problem domains. MSHH performs the best on average only on the Inst1 VRP instance, while RHH and GGHH perform better on three and one VRP instances, respectively. This appears to be an indication that application of all low level heuristics and performing local search and accepting solutions which is the best at any given time potentially a better approach on the VRP domain. DRD performs the worst on the SAT problem domain, but delivers a good average performance on the BP problem domain. GGHH and DRW manage to provide the best average results on a single instance of VRP and PFS, respectively. MSHH is better than HySST on all problem instances across all domains and this performance difference is statistically significant.

5.3.4 Performance Comparison to the Mock Competition Hyper-heuristics

The performance of the Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) is compared to the performances of eight different previous hyper-heuristics (HH1-HH8) across four problem domains, each with 10 different instances, as provided for the mock competition[2].

The problem domains used in the mock competition are Boolean Satisfiability (SAT), One Dimensional Bin Packing (BP), Personnel Scheduling (PS) and Permutation Flow Shop (PFS). A single run is performed using each problem instance in the mock competition. Table 5.10, 5.11, 5.12 and 5.13 compare the performance of the proposed multi-stage hyper-heuristic (MSHH) to the others (HH1–HH8) over a set of problem instances for SAT, BP, PS and PFS, respectively, based on the objective values obtained at the end of each run.

MSHH outperforms the mock competition hyper-heuristics with a Formula One points scoring system of 297.75 in the overall (Figure 5.14). It obtains the best results in

[2] www.asap.cs.nott.ac.uk/external/chesc2011/defaulthh.html

TABLE 5.9: The performance comparison of MSHH, GGHH, DRD, RHH, HySST and DRW multi-stage hyper-heuristics based on the average (avg.), associated standard deviation (std.), minimum (min.) of the objective values over 31 trials and the pairwise average performance comparison of MSHH vs (GGHH, DRD, RHH, HySST and DRW) based on Mann-Whitney-Wilcoxon for each CHeSC 2011 instance produced by each approach. The hyper-heuristic producing the best value for avr. and min. per each instance are highlighted in bold

		MSHH				GGHH				DRD				RHH				HySST				DRW		
Domain	Instance	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.
SAT	Inst1	**0.9**	0.7	**0.0**	>	18.4	4.5	9.0	>	27.7	5.0	16.0	>	6.6	1.2	3.0	>	3.2	1.2	1.0	>	7.4	2.7	2.0
	Inst2	**3.1**	3.9	**1.0**	>	42.0	9.8	17.0	>	58.2	8.3	27.0	>	11.8	1.9	8.0	>	16.0	20.4	2.0	>	15.2	12.2	5.0
	Inst3	**0.7**	0.5	**0.0**	>	22.0	7.8	10.0	>	40.5	6.1	17.0	>	4.3	1.5	1.0	>	4.0	6.8	1.0	>	9.4	7.9	**0.0**
	Inst4	**1.7**	1.0	**1.0**	>	19.5	3.5	10.0	>	29.9	3.7	22.0	>	8.9	2.0	4.0	>	4.2	1.5	**1.0**	>	6.3	2.0	2.0
	Inst5	**7.6**	0.9	**7.0**	>	10.4	1.5	**7.0**	>	18.5	2.7	13.0	>	8.5	0.8	**7.0**	>	9.7	2.5	**7.0**	>	10.5	1.5	**7.0**
BP	Inst1	0.0163	0.0014	0.0136	>	0.0608	0.0038	0.0541	<	**0.0133**	0.0018	**0.0109**	≥	0.0167	0.0016	0.0136	>	0.0659	0.0066	0.0536	>	0.0228	0.0022	0.0190
	Inst2	**0.0037**	0.0015	**0.0025**	>	0.0105	0.0017	0.0073	>	0.0084	0.0014	0.0071	>	0.0071	0.0010	0.0036	>	0.0158	0.0036	0.0116	>	0.0070	0.0012	0.0036
	Inst3	**0.0050**	0.0015	**0.0025**	>	0.0209	0.0018	0.0179	>	0.0135	0.0011	0.0114	>	0.0070	0.0016	0.0046	>	0.0330	0.0043	0.0238	>	0.0071	0.0011	0.0050
	Inst4	**0.1084**	0.0000	**0.1083**	>	0.1132	0.0006	0.1116	≥	0.1084	0.0000	0.1083	>	0.1085	0.0000	0.1084	>	0.1268	0.0011	0.1226	>	0.1084	0.0000	0.1084
	Inst5	**0.0050**	0.0019	**0.0032**	>	0.0379	0.0023	0.0322	>	0.0209	0.0023	0.0166	>	0.0079	0.0020	0.0053	>	0.0632	0.0048	0.0542	>	0.0074	0.0022	0.0034
PS	Inst1	**25.5**	4.5	**16.0**	>	32.5	5.2	23.0	≥	27.0	4.1	20.0	≥	27.4	4.1	20.0	>	49.6	4.1	41.0	>	30.6	4.0	24.0
	Inst2	9668.9	217.8	**9184.0**	≥	9695.8	135.0	9415.0	≥	9684.7	110.3	9433.0	≤	**9644.6**	114.6	9405.0	>	10131.0	140.5	9905.0	≥	9723.3	147.7	9486.0
	Inst3	**3283.7**	93.3	**3132.0**	≥	3303.8	93.3	3182.0	≥	3321.0	112.3	3157.0	≥	3316.9	88.4	3158.0	>	3669.5	130.3	3461.0	≥	3292.8	87.5	3139.0
	Inst4	1786.3	172.1	1545.0	≥	1791.9	217.7	1505.0	≤	1773.5	173.8	1523.0	≤	**1738.0**	99.3	1555.0	>	58241.9	7050.4	47646.0	≤	1754.7	146.2	**1448.0**
	Inst5	**353.2**	21.2	**315.0**	>	696.9	519.6	340.0	≥	395.7	235.3	325.0	>	385.3	30.6	330.0	>	208250.2	10796.2	192036.0	>	769.0	838.7	355.0
PFS	Inst1	**6239.8**	14.9	**6212.0**	>	6302.8	11.3	6275.0	>	6333.1	22.5	6285.0	>	6285.3	8.8	6267.0	>	6341.2	25.5	6300.0	>	6255.8	14.3	6228.0
	Inst2	26895.2	55.3	26775.0	≤	26872.6	32.5	26805.0	>	26963.5	56.9	26860.0	>	26931.3	41.0	26846.0	>	27101.0	17.0	27068.0	<	**26807.6**	36.0	**26735.0**
	Inst3	**6333.8**	19.0	**6303.0**	>	6368.3	3.9	6352.0	>	6395.0	19.4	6359.0	>	6354.4	13.1	6326.0	>	6395.4	17.5	6369.0	>	6349.6	22.7	**6303.0**
	Inst4	**11363.8**	32.7	11320.0	>	11454.4	19.4	11420.0	>	11505.3	27.8	11455.0	>	11455.3	19.5	11415.0	>	11551.1	23.1	11482.0	≥	11380.8	34.6	**11319.0**
	Inst5	26711.9	47.0	26630.0	<	26682.0	27.0	26624.0	>	26756.3	54.2	26639.0	≥	26733.9	35.7	26632.0	>	26882.4	18.2	26841.0	<	**26644.0**	29.2	**26598.0**
TSP	Inst1	**48208.1**	31.8	**48194.9**	>	49189.7	280.2	48658.0	>	49561.2	513.7	48579.8	>	49946.2	354.9	48862.2	>	50633.4	2321.2	49356.2	>	50071.6	579.1	49028.9
	Inst2	**2.09e^{+7}**	9.05e^{+4}	**2.07e^{+7}**	>	2.14e^{+7}	1.60e^{+5}	2.12e^{+7}	>	2.13e^{+7}	1.05e^{+5}	2.11e^{+7}	>	2.32e^{+7}	1.82e^{+6}	2.13e^{+7}	>	2.49e^{+7}	1.20e^{+5}	2.47e^{+7}	>	2.13e^{+7}	8.91e^{+4}	2.11e^{+7}
	Inst3	**6809.1**	7.1	**6796.6**	>	7008.8	22.3	6928.2	>	7052.0	31.9	6993.6	>	7063.1	22.5	7013.8	>	8093.3	88.8	7934.9	>	7023.7	41.6	6963.6
	Inst4	**66840.2**	276.5	**66236.8**	>	71115.5	1245.2	69138.4	>	70161.9	445.4	69397.5	>	72505.3	2540.5	69036.3	>	78243.2	739.8	76803.1	>	68817.0	394.5	68088.7
	Inst5	**53011.4**	469.7	**52341.3**	>	57995.5	1643.8	54137.8	>	55144.2	744.3	53787.7	>	57138.0	2417.3	54477.8	>	60433.8	755.5	59470.7	>	54427.7	621.9	53601.3
VRP	Inst1	**70998.4**	3840.3	**63948.2**	>	81271.4	4660.9	73486.9	≥	71768.6	4168.4	65292.3	≥	71146.5	3673.4	65726.4	>	99928.7	2677.2	95327.9	>	97991.1	3940.5	91978.2
	Inst2	13421.8	251.6	13303.9	<	13418.5	41.3	13319.7	>	14523.1	680.4	13354.4	≤	**13405.4**	183.5	13327.0	>	13832.4	605.9	**12351.7**	>	14015.3	601.4	13367.0
	Inst3	148498.2	1625.8	145466.5	>	171937.4	3951.0	163675.7	>	151904.0	1871.8	148551.8	<	**146174.9**	1817.0	**142480.2**	>	350453.2	6010.8	336025.8	>	307301.2	15683.8	280087.2
	Inst4	21016.4	488.2	**20650.8**	≤	**20669.7**	9.7	20654.6	>	21910.4	564.5	20657.2	≤	20766.5	295.9	20653.8	>	24370.2	903.8	21048.8	>	22072.5	560.5	20827.7
	Inst5	148813.7	1272.5	146334.6	>	153931.3	1621.4	150687.8	>	149886.1	1382.4	147255.8	<	**147234.4**	622.8	**145552.5**	>	208923.3	5196.0	202032.9	>	196210.4	6374.2	183749.7

22 out of 40 (55%) instances with 6 draws mostly in the SAT and 1D Bin Packing problems. In the personnel scheduling problem, MSHH produces the best results in 1 instance. It provides the best results in 3 instances and a tie in PFS problem domain. MSHH is the winner in the SAT, BP and PFS problem domains and looses to the other hyper-heuristics in the PS problem domain.

TABLE 5.10: SAT, objective function values obtained by the eight hyper-heuristics and MSHH on the 10 instances. The last row summarises the number of wins/draws. The best values for each instance are highlighted in bold

Instance	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	MSHH
Inst1	46	33	14	28	119	12	56	40	**6**
Inst2	40	33	36	50	136	34	38	66	**23**
Inst3	32	**24**	28	47	116	29	35	53	**24**
Inst4	16	13	35	24	60	15	15	25	**1**
Inst5	9	10	45	37	70	33	9	36	**1**
Inst6	22	17	52	52	106	51	24	55	**3**
Inst7	6	6	8	12	18	9	**5**	15	**5**
Inst8	**6**	**6**	8	11	13	11	**6**	14	**6**
Inst9	8	**7**	11	16	21	12	9	19	**7**
Inst10	**211**	**211**	221	239	259	215	217	239	**211**
w/d	0/2	0/4	-	-	-	-	0/2	-	**5/5**

TABLE 5.11: BP, objective function values obtained by the eight hyper-heuristics and MSHH on the 10 instances. The last row summarises the number of wins/draws. The best values for each instance are highlighted in bold

Instance	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	MSHH
Inst1	0.0174	0.0176	0.0108	0.0120	0.0541	0.0157	0.0217	0.0714	**0.0028**
Inst2	0.0163	0.0165	0.0071	0.0077	0.0501	0.0129	0.0214	0.0712	**0.0067**
Inst3	0.0238	0.0229	0.0247	0.0230	0.0283	0.0231	0.0236	0.0308	**0.0210**
Inst4	0.0248	0.0249	0.0266	0.0243	0.0308	0.0257	0.0255	0.0327	**0.0190**
Inst5	0.0064	0.0062	**0.0003**	0.0046	0.0151	0.0066	0.0073	0.0218	0.0048
Inst6	0.0040	0.0085	0.0036	0.0036	0.0187	0.0089	0.0090	0.0234	**0.0031**
Inst7	0.1145	0.1047	**0.0107**	0.0312	0.1715	0.0538	0.1386	0.1666	0.0161
Inst8	0.1337	0.1285	**0.0168**	0.0640	0.1719	0.0942	0.1506	0.1799	0.0216
Inst9	0.0559	0.0569	0.0562	0.0944	0.0927	0.0608	0.0582	0.1267	**0.0457**
Inst10	0.0135	0.0128	0.0190	0.0309	0.0344	0.0166	0.0139	0.0428	**0.0036**
w/d	-	-	3/0	-	-	-	-	-	**7/0**

TABLE 5.12: PS, objective function values obtained by the eight hyper-heuristics and MSHH on the 10 instances. The last row summarises the number of wins/draws. The best values for each instance are highlighted in bold

Instance	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	MSHH
Inst1	3346	3321	3389	**3318**	3338	8017	3344	3342	3325
Inst2	2220	2315	2400	2275	2454	21008	**2095**	2893	2609
Inst3	390	400	495	375	400	905	355	**340**	420
Inst4	23	17	32	19	**16**	80	22	**16**	21
Inst5	23	26	32	24	24	81	**19**	28	26
Inst6	**17**	**17**	32	24	22	81	28	38	29
Inst7	**1111**	1119	1231	1118	1113	35391	1211	1490	1284
Inst8	**2188**	2202	2205	2221	2288	46661	2275	3959	2297
Inst9	**3163**	3255	3465	3360	3354	46952	3414	6905	3430
Inst10	11486	9706	12505	12994	9771	105850	9807	17224	**9509**
w/d	**3/1**	0/1	-	1/0	0/1	-	2/0	1/1	1/0

TABLE 5.13: PFS, objective function values obtained by the eight hyper-heuristics and MSHH on the 10 instances. The last row summarises the number of wins/draws. The best values for each instance are highlighted in bold

Instance	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	MSHH
Inst1	6365	6380	6399	6312	6393	6297	6375	6323	**6279**
Inst2	6327	6330	6337	6281	6328	**6253**	6335	6288	6258
Inst3	6401	6410	6401	6339	6418	6339	6407	6364	**6324**
Inst4	6388	6408	6366	6327	6373	6366	6371	6363	**6325**
Inst5	6461	6470	6438	**6392**	6483	6405	6478	6422	6410
Inst6	10540	10546	10506	**10499**	10547	10509	10546	10542	10501
Inst7	10976	10965	10965	**10923**	10980	**10923**	10965	10956	**10923**
Inst8	26483	26490	26538	26409	26506	26418	26512	**26396**	26492
Inst9	26979	26929	26978	26890	26913	26920	26960	**26800**	26824
Inst10	26755	26794	26833	26731	26755	**26715**	26811	26716	26754
w/d	-	-	-	2/1	-	2/1	-	2/0	**3/1**

5.3.5 Performance Comparison to the CHeSC 2011 Hyper-heuristics

MSHH and the twenty competing hyper-heuristics from CHeSC 2011 are ranked under the same criteria used at the time of the competition. The state-of-the-art hyper-heuristic, denoted as *AdapHH* is a hyper-heuristic which combines a learning adaptive heuristic selection method, that identifies poorly performing low level heuristics and discards them during the search process, with an adaptive iteration limited list-based threshold move accepting method [27]. AdapHH can be considered as a multi-stage hyper-heuristic approach, managing two hyper-heuristics. One hyper-heuristic aims to

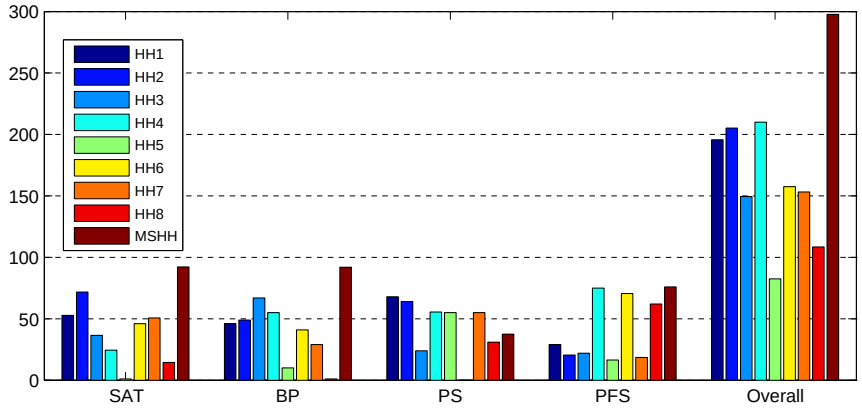


FIGURE 5.14: Comparisons of the different hyper-heuristics over each domain based on Formula One points scoring system

exclude the relatively poor performing low level heuristics including the relay hybridised pairs of heuristics. The second hyper-heuristic uses the information provided by the first hyper-heuristic and combines a roulette wheel selection for choosing a heuristic with an adaptive move acceptance method. AdapHH introduces over 45 parameters which are either already set or require control. Moreover, the number of lines of the Java code for the implementation of the method is over 3000[3]. We believe that our approach (MSHH) is easier to implement requiring setting of only six parameters and less than 300 lines of code.

Table 5.14 presents the scores for each algorithm based on the Formula One scoring system with respect to the median objective values obtained during the 31 trials over all instances across the six domains. Although MSHH delivers a relatively poor "median" performance in the PS and VRP problem domains, the overall results reveal that MSHH is the winner with a total score of 163.60, performing better than AdapHH in the overall. In general, the heuristic selection and move acceptance components of MSHH interacts well yielding a better performance with respect to AdapHH. Another reason for the success of MSHH against AdapHH could be that the adaptive iteration limited list-based threshold move accepting method used in AdapHH allows diversification (exploration) only if the intensification (exploitation) phase does not yield any improvements for a pre-defined number of steps. Our move acceptance method potentially allows the transition from intensification to diversification quicker than AdapHH discovering potentially

[3] `http://code.google.com/p/generic-intelligent-hyper-heuristic/`

better regions of the search space leading to higher quality solutions. This observation seems to hold only for some problem domains, such as SAT and TSP.

TABLE 5.14: Ranking (performance comparison) of MSHH and the 20 hyper-heuristic approaches competed at CHeSC 2011 across six problem domains based on the Formula One scoring system

Label	SAT	BP	PS	PFS	TSP	VRP	Overall
MSHH	48.00	38.00	6.00	25.00	42.60	4.00	**163.60**
AdapHH	27.58	44.00	8.00	33.00	34.60	14.00	161.18
VNS-TW	27.08	2.00	39.50	30.00	13.60	6.00	118.18
ML	10.00	8.00	31.00	36.50	10.00	22.00	117.50
PHUNTER	7.00	2.00	11.50	6.00	21.60	33.00	81.10
EPH	0.00	6.00	10.50	18.00	30.60	12.00	77.10
HAHA	25.58	0.00	24.50	2.83	0.00	14.00	66.92
NAHH	10.50	16.00	2.00	19.50	9.00	6.00	63.00
ISEA	3.50	25.00	14.50	3.50	7.00	4.00	57.50
KSATS-HH	19.00	7.00	8.50	0.00	0.00	22.00	56.50
HAEA	0.00	1.00	1.00	7.33	8.00	27.00	44.33
GenHive	0.00	10.00	6.50	7.00	2.00	6.00	31.50
ACO-HH	0.00	17.00	0.00	6.33	6.00	1.00	30.33
SA-ILS	0.25	0.00	18.50	0.00	0.00	4.00	22.75
AVEG-Nep	9.50	0.00	0.00	0.00	0.00	9.00	18.50
XCJ	3.50	10.00	0.00	0.00	0.00	5.00	18.50
DynILS	0.00	9.00	0.00	0.00	8.00	0.00	17.00
GISS	0.25	0.00	10.00	0.00	0.00	6.00	16.25
SelfSearch	0.00	0.00	3.00	0.00	2.00	0.00	5.00
MCHH-S	3.25	0.00	0.00	0.00	0.00	0.00	3.25
Ant-Q	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Another performance metric is suggested in [80] to give an indication of the relative variation of each competing hyper-heuristic for each problem domain and to evaluate and compare the performance of the hyper-heuristics. The median objective function values of the 31 trials are normalised to a value in [0,1] according to Equation 5.1.

$$norm(x,i) = \frac{x(i) - x_{best}(i)}{x_{worst}(i) - x_{best}(i)} \tag{5.1}$$

where $x(i)$ is the objective function value on instance i, $x_{best}(i)$ is the best objective function value obtained by the different methods on instance i and $x_{worst}(i)$ is the worst objective function value obtained by the different methods on instance i.

Figures 5.15 and 5.16 provide the box plots of the normalised values for the MSHH and the competitors' hyper-heuristics for each domain and in overall, respectively. It

is observed that the MSHH outperforms the other approaches in overall and in SAT, PFS, and TSP problem domains and taking the second place in BP problem domain. However, the proposed hyper-heuristic delivers a relatively poor performance on the PS and VRP problem domains.

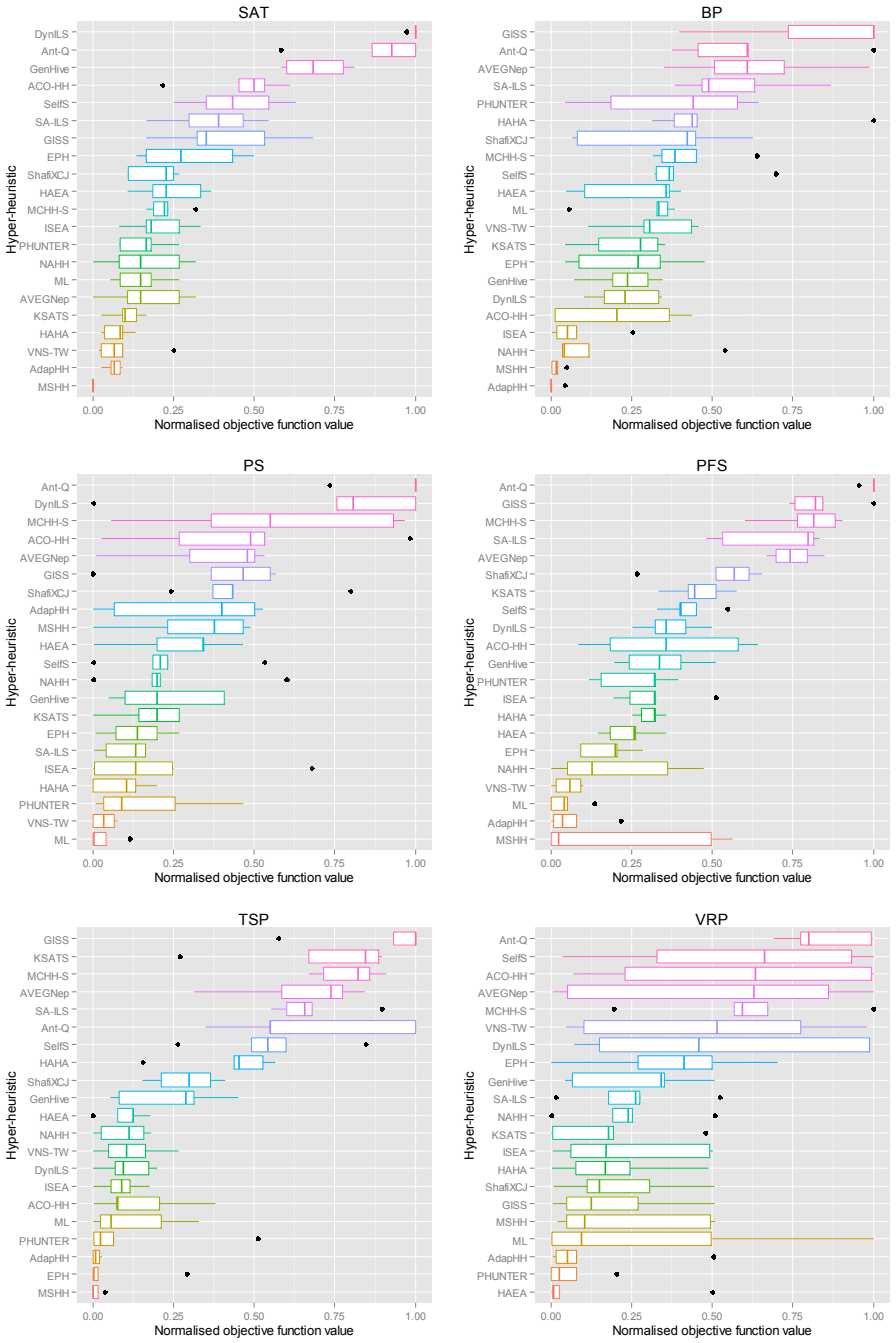


FIGURE 5.15: Ranking (performance comparison) of MSHH and CHeSC 2011 hyper-heuristics for each HyFlex problem domain based on the median results converted to the normalised objective function values. The dots in the box plots are outliers

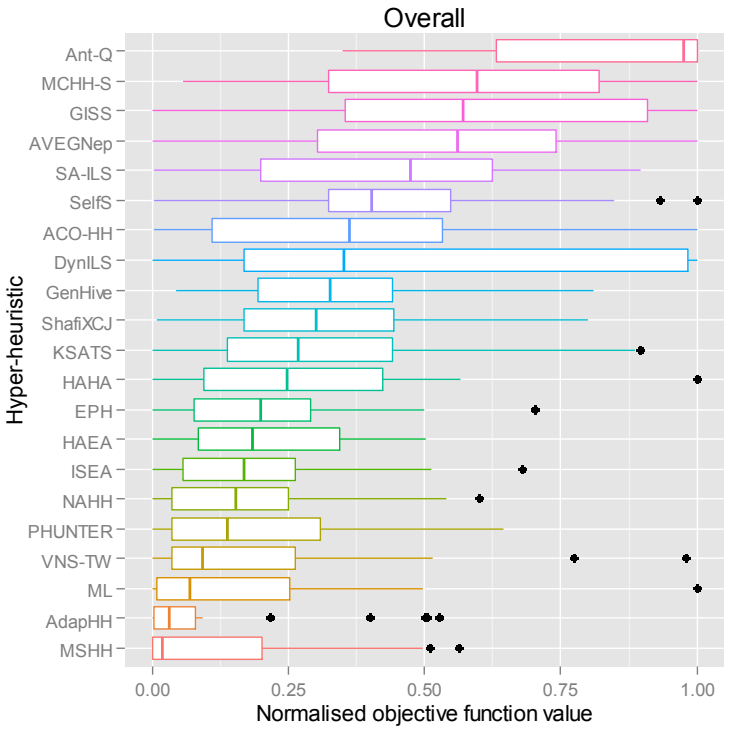


FIGURE 5.16: Ranking (performance comparison) of MSHH and CHeSC 2011 hyper-heuristics in overall based on the median results converted to the normalised objective function values. The dots in the box plots are outliers

5.3.6 An Analysis of the Proposed Hyper-heuristic

We have repeated some experiments in order to track and interpret the behaviour of MSHH. Each trial is repeated for 10 times during this set of experiments. The *percentage utilisation* is the ratio of the number of improvements that a low level heuristic generates over the best solution found so far to the total number of such improvements. Figure 5.17 shows the average percentage utilisation of the single and combined low level heuristics while an arbitrarily chosen representative instance from each problem domain is solved. As one would expect, not all the low level heuristics can generate improvement over the best solution found so far during the search process. For example, in PS, surprisingly, LLH0 and LLH1 heuristics which are provided as hill climbers do not yield any improvement neither themselves individually nor in combination with another low level heuristic on the tested instance. On the other hand, LLH3 and LLH5 are not able to make any improvement on the best solutions while the BP instance is being solved. This is not surprising, though, as those low level heuristics are mutational heuristics.

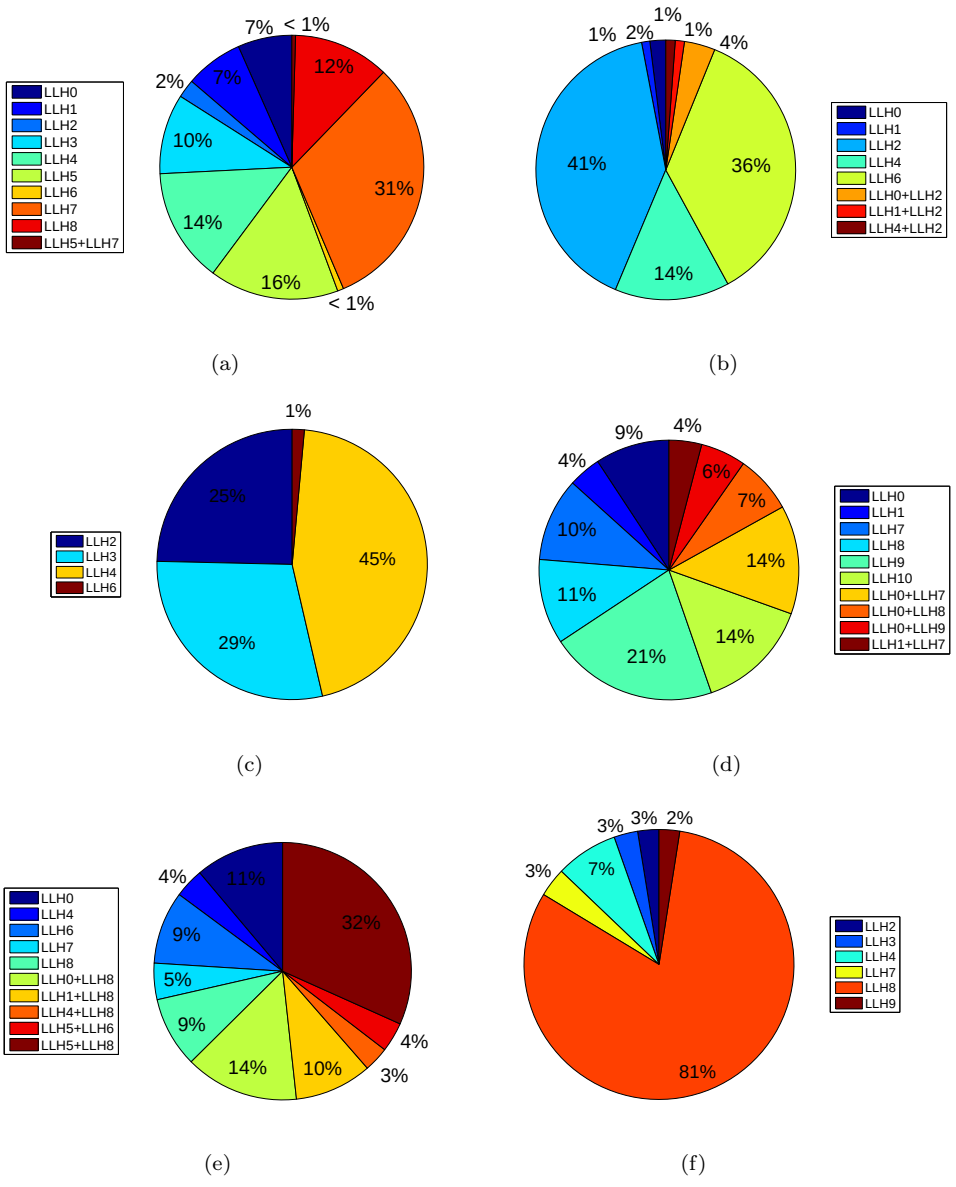
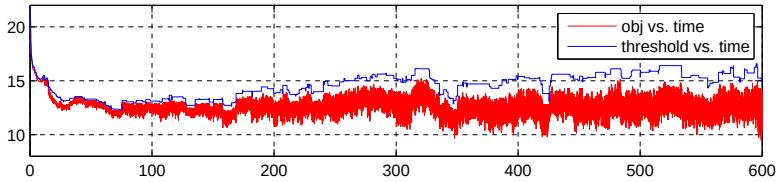


FIGURE 5.17: Average percentage utilisation of single/combined low level heuristics over 10 trials while solving a sample instance representing each problem domain: (a) SAT, (b) BP, (c) PS, (d) PFS, (e) TSP, (f) VRP

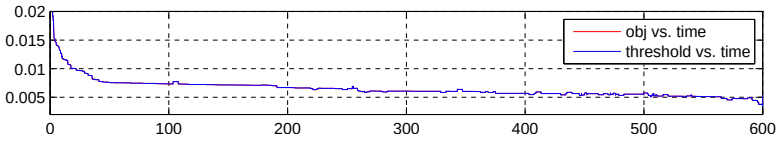
With the exception on SAT problem domain, most of the improving moves are due to hill climbers rather than mutational heuristics. The use of the combination of a mutational heuristic followed by a hill climbing heuristic, like the basic steps of iterated local search [60], is automatically favoured by our hyper-heuristic in the PFS and TSP domains (Figure 5.17(d), (e)). Similarly, ruin and re-create followed by a hill climber is another favourite automatically detected pairing in the TSP problem domain. In TSP, LLH1 does not seem to be that useful at the first glance, but considering the relay hybridisation technique, it seems to serve as a 'good' diversification component, improving the performance of the hill climbing heuristic (LLH8) employed afterwards. In BP, MSHH favours the pairing of a mutational low level heuristic followed by a ruin and re-create heuristic. The relay hybridisation of low level heuristics seem to be useful, except for the PS and VRP domains, in which it has been observed that no generated heuristic pairs contributed towards the improvement of the best solutions. The proposed hyper-heuristic looses time by testing all pairs of given low level heuristics which could have been used in the search process. This could be one of the reasons why the proposed hyper-heuristic performs relatively poor on those domains.

The behaviour of MSHH considering the average threshold value of the move acceptance method and average objective values of the current solution in time is illustrated in Figure 5.18 for an arbitrarily selected instance from each problem domain. In some cases, MSHH improves the quality of the initial solution at the beginning of the search process rapidly. Then the improvement slows down, but still continues as in the BP and VRP domains (Figure 5.18(b), (f)). While MSHH solves a given instance, it enters into what seems to be a "neural" region getting stuck at a local optimum for a while. Then, MSHH is able to find a way to make improvement (e.g., Figure 5.18(c), (e)). In the overall, the adaptive move acceptance method successfully supports further improvements by allowing worsening solutions at different parts of the search process. MSHH seems to require partial restarts while solving problem instances from the SAT and PFS problem domains more than the others which definitely works and this could be one of the reasons for the success of MSHH on those problem domains.

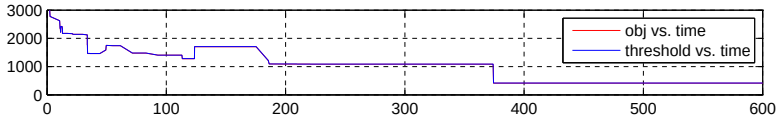
Figure 5.19 depicts the progress of the average number of low level heuristics including individual and paired low level heuristics used in time over 10 trials for each problem domain on a selected instance. Interestingly, on average, approximately less than 10% of the low level heuristics are used for each problem across six problem domains. Stage two hyper-heuristic ignores most of the low level heuristics including the ones generated as pairs from the relay hybridisation process as illustrated. A change in that value occurs



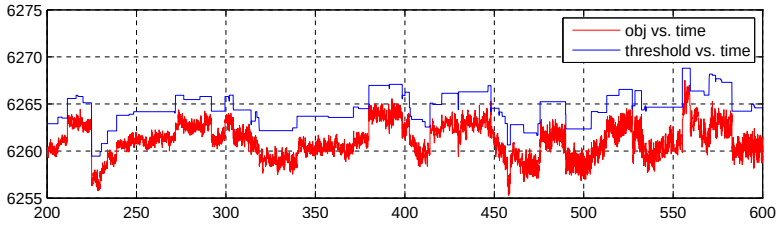
(a) SAT problem domain



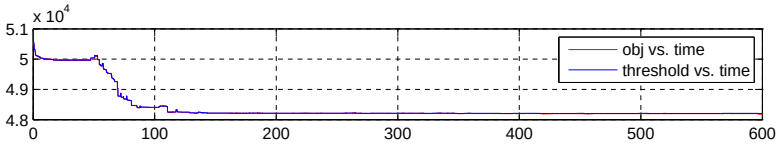
(b) BP problem domain



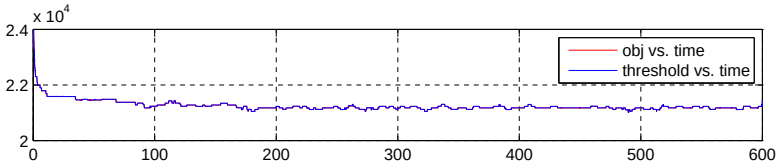
(c) PS problem domain



(d) PFS problem domain



(e) TSP problem domain



(f) VRP problem domain

FIGURE 5.18: Plots of the average objective and threshold level values over 10 trials versus time while solving a sample instance representing each problem domain

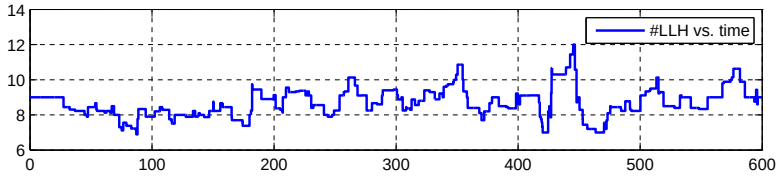
after the multi-stage level applies the second stage hyper-heuristic. In PS problem domain, all the single low level heuristics are used and so the whole set of heuristics is needed during the entire search process. The fluctuations in the number of low level heuristics used during the search process are very frequent in all the other problem domains. It has been observed that the number of low level heuristics never decreases to a single low level heuristic at any time in none of the domains. Figure 5.19 illustrates that different sets of low level heuristics are useful at different parts of the overall search process. For example, at the start of the search process, the number of the low level heuristics stays the same for BP, then it starts decreasing towards the midst of the given time.

5.4 Experimental Results on Constructing Magic Square Problem

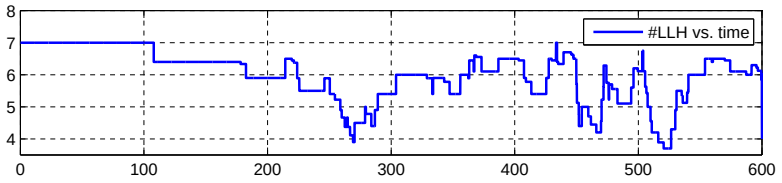
All computational experiments are performed on small instances from n=10 up to 23 with increments of 1 and large instances from n=25, 50, 75, 100 up to 2600 with increments of 100, unless mentioned otherwise. 2600 is chosen as the maximum order for the magic squares problem, as the winning approach of the magic square competition was able to solve a magic squares problem of order 2600 as the largest instance under a minute on the competition computer. Since the specification of the competition computer is not known, we performed our experiments on an i3 CPU M330 at 2.13GHz with a memory of 4.00GB and each one is repeated for 50 trials. A trial is terminated, as soon as a solution is found under one minute on our computer. The placement of the upper left-hand corner of the sub-matrix $S_{3\times3}$ within the main matrix has been arbitrarily selected to be at the position (1,4). A final set of experiments are performed for some n, using different random locations. Unlike previous studies on hyper-heuristics, the performance of an approach is measured with its run-time rather than the quality of solutions obtained for the given problems.

5.4.1 Comparison of MSHH to the Best Known Heuristic Approaches

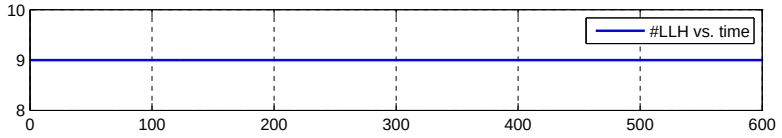
All approaches are tested with the goal of detecting the quickest one. Table 5.15 summarises the performance comparison of MSHH to the best previously proposed solution methodologies (LAHC and RP) which are the winner of the magic squares competition and the quickest-known approach, respectively, on some selected instances of order n.



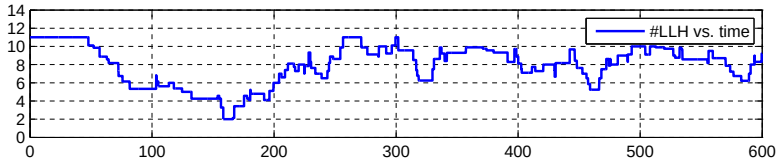
(a) SAT problem domain



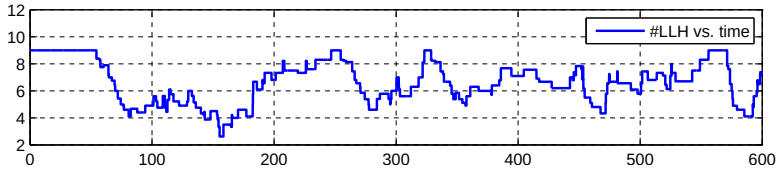
(b) BP problem domain



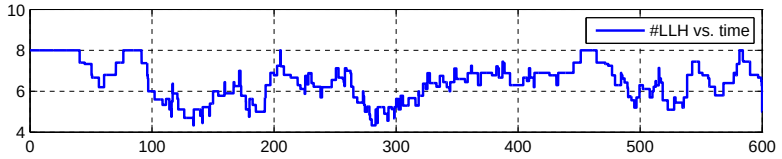
(c) PS problem domain



(d) PFS problem domain



(e) TSP problem domain



(f) VRP problem domain

FIGURE 5.19: Plots of the average of changes in the number of single/combined low level heuristics versus time from 10 trials while solving a sample instance representing each problem domain

The table provides the average execution time, the standard deviation and the pairwise performance comparison in millisecond over 50 trials of arbitrarily chosen 10 sample instances from small and large orders of n for each. The MSHH performs the best on two instances. Overall, RP seems to perform better than the other approaches on average. MSHH consistently performs better than LAHC for all instances, except $n = 2600$, on average. LAHC performs worse than the other two approaches in most of the instances and this performance difference is statistically significant.

TABLE 5.15: The average execution time (avr.) the standard deviation (s.d.) in milliseconds and the pairwise performance comparison of 50 trials. The best values are highlighted in bold

	MSHH		LAHC		RP		MSHH vs	MSHH vs	LAHC vs
n	avg.	std.	avg.	std.	avg.	std.	LAHC	RP	RP
10	**249**	215	3825	3221	250	456	$>$	$\geq$	$<$
11	249	335	3409	4070	**164**	142	$>$	$\leq$	$<$
13	312	240	4823	4595	**241**	160	$>$	$\leq$	$<$
14	354	294	7841	8284	**327**	250	$>$	$\leq$	$<$
15	473	393	7026	5603	**308**	231	$>$	$<$	$<$
16	552	476	8356	8106	**397**	313	$>$	$<$	$<$
18	611	392	8268	5905	**684**	632	$>$	$\leq$	$<$
19	1021	1174	11325	10572	**659**	461	$>$	$<$	$<$
21	1133	707	16061	12340	**819**	657	$>$	$<$	$<$
23	1624	1760	27399	25735	**1446**	1256	$>$	$\leq$	$<$
25	**10**	7	157	26	14	13	$>$	$\geq$	$<$
50	42	22	366	252	**39**	28	$>$	$\leq$	$<$
100	59	31	415	351	**56**	49	$>$	$\leq$	$<$
200	164	112	1249	1140	**113**	85	$>$	$<$	$<$
400	524	468	1790	1498	**260**	188	$>$	$<$	$<$
800	1337	1125	3960	2722	**556**	290	$>$	$<$	$<$
1000	2077	3597	4620	2775	**692**	464	$>$	$<$	$<$
1500	5407	5459	5676	3957	**1252**	649	$\geq$	$<$	$<$
2000	5938	5177	6161	3822	**2036**	881	$\geq$	$<$	$<$
2600	11260	9870	8142	4971	**3684**	1559	$\leq$	$<$	$<$

The inclusion of multiple low level heuristics and the stochastic nature of the hyper-heuristic makes it extremely difficult to compute the running time complexity of the overall algorithm. Hence, a regression model is formed based on large n. 50 trials to construct magic square of various orders have been considered for the regression model. Table 5.16 provides the Root Mean Square Error (RMSE) to indicate the quality of the fit. All the three approaches run in $O(n)$ time. The constant multiplier is almost similar in both LAHC and MSHH while RP has a smallest constant coefficient and RMSE values, showing that RP runs predictably faster than MSHH and LAHC.

TABLE 5.16: Regression models to predict the running time complexity of the MSHH, LAHC and RP approaches

Apprach	Model	Multiplier	RMSE
MSHH	$a \cdot n$	$a = 3.5$	4202
LAHC	$a \cdot n$	$a = 3.4$	2780
RP	$a \cdot n$	$a = 1.1$	712

To test the importance of separating the set of low level heuristics, three different regression models are tested by applying only the first set of low level heuristics. The regression sum of squares (RSS) from these models show that the hyper-heuristic approach executes in $O(n^4)$ second as illustrated in Figure 5.20, which is way worse than the results presented in Table 5.16.

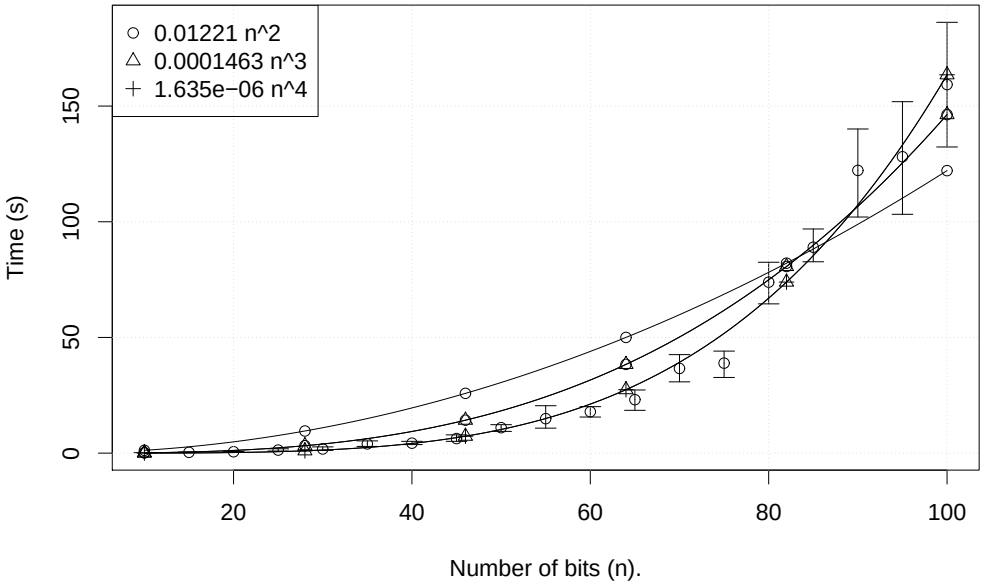


FIGURE 5.20: The box plot of execution time (in seconds) of the hyper-heuristic approach constructing a magic square of various orders, n and plots of the regression models

A final set of experiments are performed to observe the behaviour of MSHH, LAHC and RP approaches for the instances of orders n=10, 23, 25 and 2600 varying the placement of the upper left-hand corner of the sub-matrix $S_{3\times3}$ at (i, j). The selected instance are the smallest and the largest orders of both small and large sets. We generated 200 random locations of (i, j) for small n=10, 23 and large n=25, 2600 instances. Figure 5.21 provides the box plots obtained from MSHH, LAHC and RP for their running times

showing that MSHH clearly outperforms the LAHC approach for instances of n=10, 23 and 25 and performs slightly better on average than the RP approach.

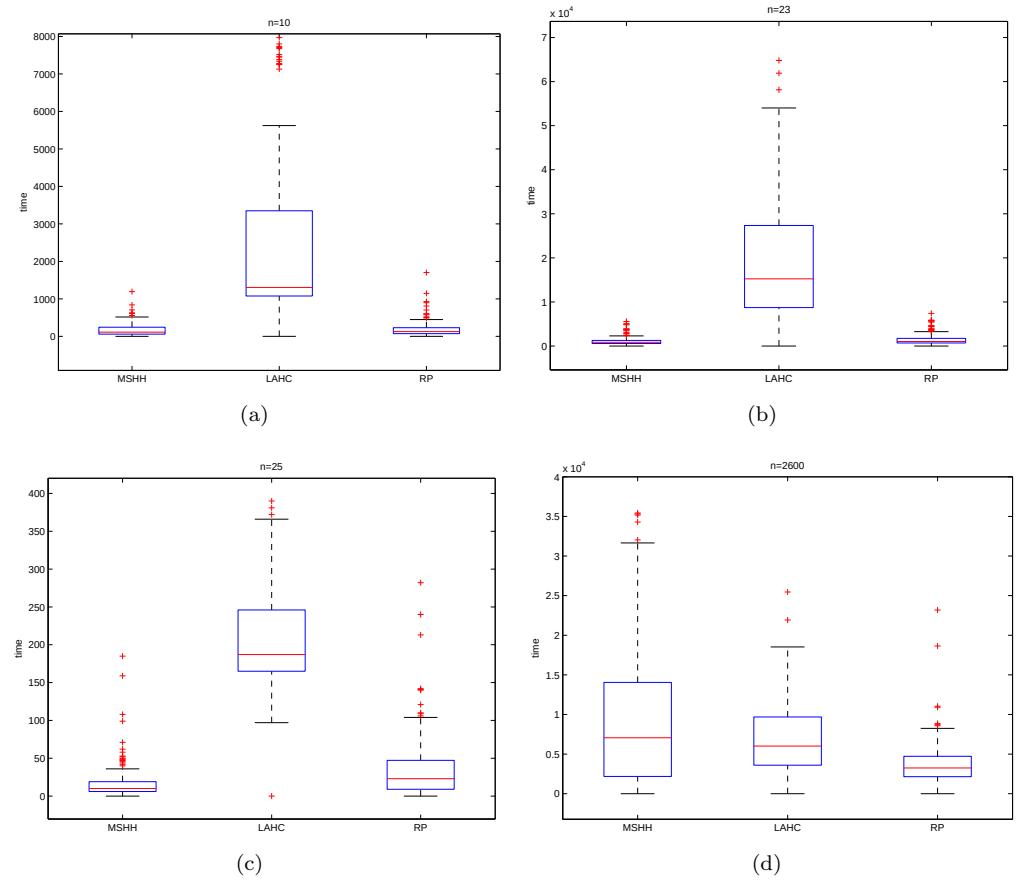


FIGURE 5.21: Box plots of execution time (in milliseconds) from all runs for MSHH, LAHC and RP approaches constructing a constrained magic square using various randomly decided (i, j) locations for n = (a) 10, (b) 23, (c) 25 and (d) 2600

5.4.2 Performance Analysis of the Multi-stage Hyper-heuristic

Different low level heuristics contribute to the improvement of a solution in hand at different levels. Figure 5.22 provides the average percentage utilisation of 10 trials of each low level heuristic considering improving moves only using a sample run for n=10, 23, 25 and 2600. In the first set of low level heuristics, LLH0 and LLH3 are more successful with high utilisation rates in improving a candidate solution as compared to others, in general. Similarly, LLH1, LLH2, LLH5, LLH6 and LLH8 perform better than the rest in this respect. In the second set of low level heuristics, it is observed that LLH1 generates more improving moves as compared to LLH0 when n=25, while the situation

is vice verse in n=2600. Almost in all cases, no pair of heuristics contributed to the solving of the problem.

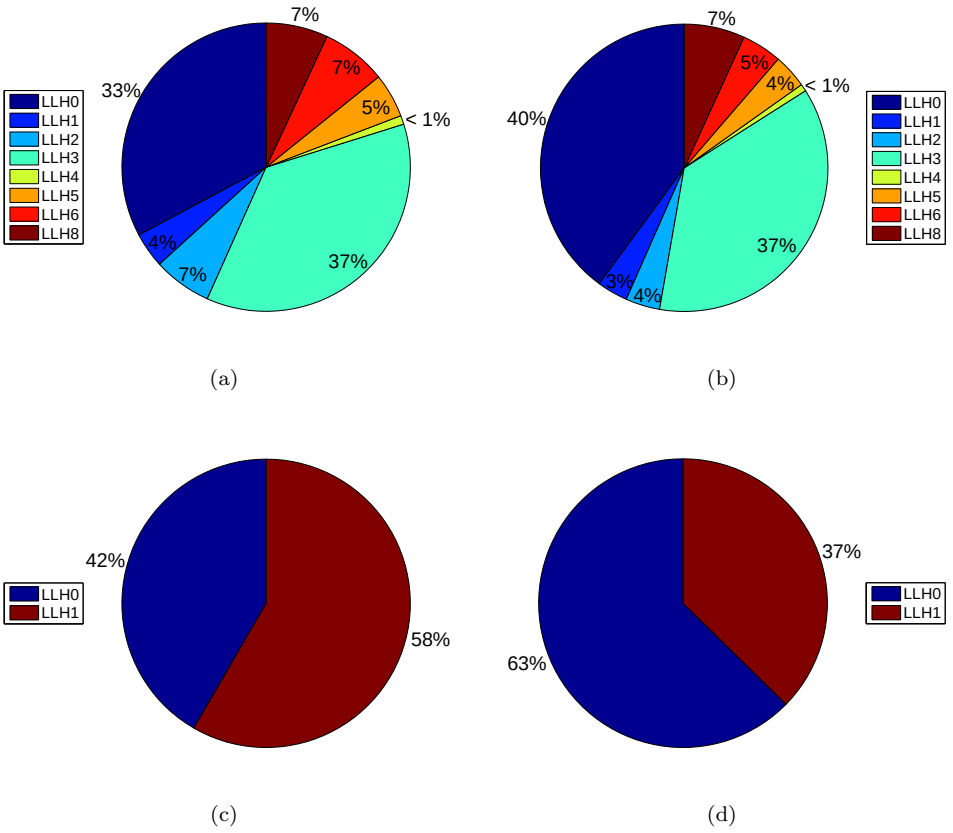


FIGURE 5.22: Average percentage utilisation of the low level heuristics obtained from 10 trials based on improving moves only for n = (a) 10, (b) 23, (c) 25 and (d) 2600

We have investigated the behaviour of MSHH based on the proposed acceptance method. In most of the cases, the MSHH rapidly improves the quality of the solution in hand. After a while, the improvement process slows down as the approach reaches a local optimum. Still, it seems that the threshold acceptance method works well as a part of the proposed approach, allowing further improvement in time even if takes a while to obtain a magic square of the given order. The proposed move acceptance allows partial restarts and the extension of these restarts changes if there is no improvement and in general there is some improvement. This behaviour is illustrated in Figure 5.23 for n=10, 23, 25 and 2600. The designed multi-stage solver is fast-enough to solve the problem without requiring of applying the S2HH in almost all the cases. This observation is typical to the previous findings in [166] such that learning requires time slowing down

a selection hyper-heuristic and so hyper-heuristics with no learning method are more successful than the learning hyper-heuristics in solving this particular problem.

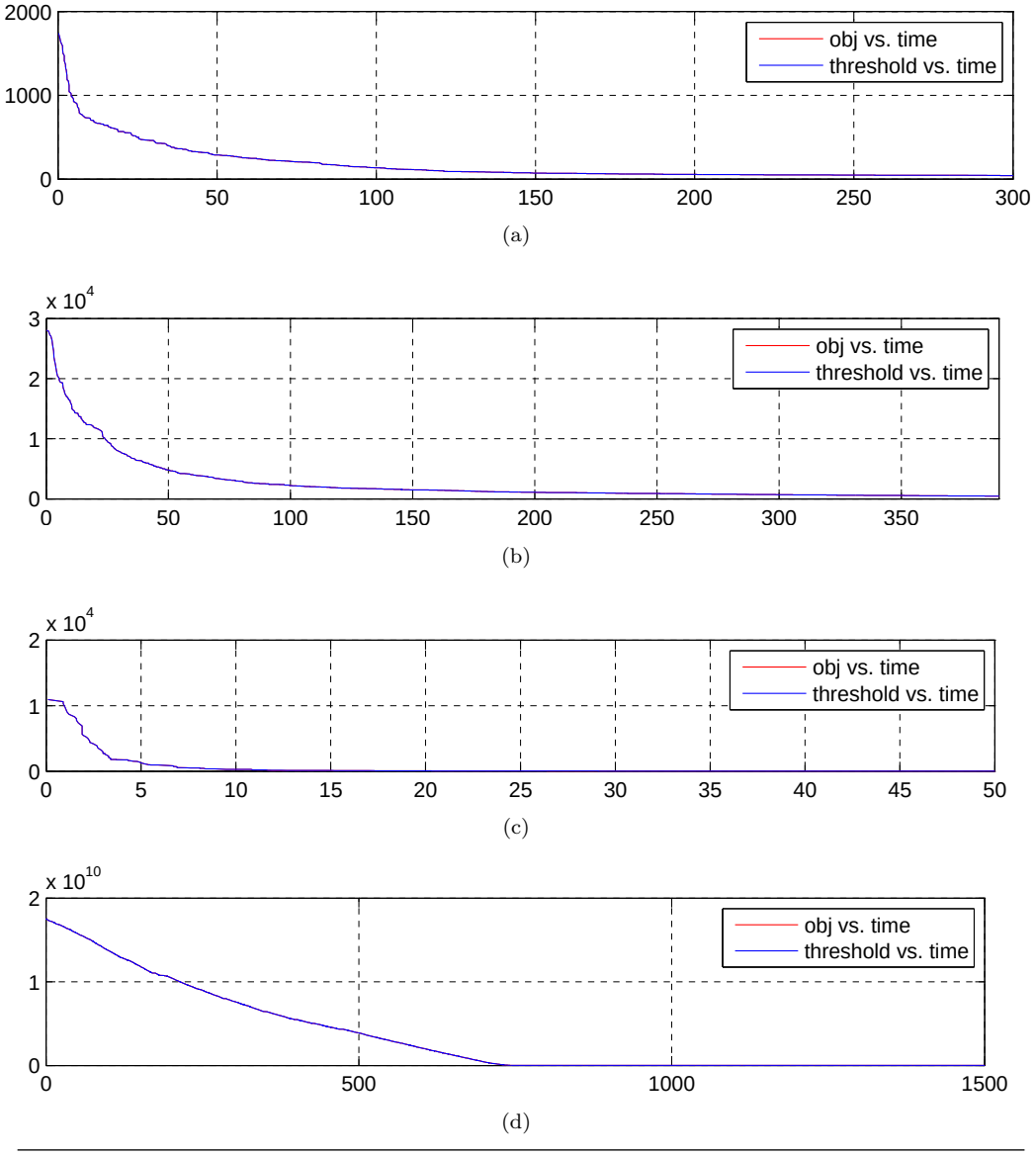


FIGURE 5.23: Plots of the average of objective values and the threshold level values versus time in milliseconds from 10 trials for n = (a) 10, (b) 23, (c) 25 and (d) 2600

5.5 Summary

For empirically examining the performance of the multi-stage hyper-heuristics and to compare their performance to the state-of-the-art, we have participated in ITC 2011 and

MISTA 2013 competitions. We joined ITC 2011 as the team HySST (Hyper-heuristic Search Strategies and Timetabling). Despite this being the first attempt in high school timetabling, HySST generated the best new solutions for three given instances in Round 1 and gained the second place in Rounds 2 and 3 with a fairly standard stochastic search method but significantly enhanced by a multi-stage hyper-heuristic with an adaptive acceptance mechanism. The Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance (DRW) multi-stage hyper-heuristic is used as a component in an effective hybrid approach which won the MISTA 2013 challenge with a mean rank of 1.1 for multi-mode resource-constrained multi-project scheduling problem. A basic performance analysis of the proposed approach is also provided in this chapter. The success provides evidence of the utility of carefully designed and controlled hybrids of mixes of (mostly) pre-existing search concepts.

The proposed learning Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) is tested on a benchmark of problem domains. The results confirm its success when compared to each constituent hyper-heuristics, the other proposed multi-stage hyper-heuristics as well as the state-of-the-art hyper-heuristic which won the CHeSC 2011 competition. The approach is a relatively simple approach which is easy-to-implement and easy-to-maintain as compared to some previously proposed hyper-heuristics including the previous state-of-the-art hyper-heuristic, yet, it is extremely effective in cross-domain search delivering a superior performance.

The MSHH is tested in solving the constrained version of magic squares and compared to the winner approach of the competition organised by SolveIT Software known as LAHC, and also compared to RP (random permutation hyper-heuristic) which is the best and the quickest-known approach in solving the constrained version of magic squares as reported in [166]. The MSHH performs the best on some instances but overall, RP performs better on average. MSHH consistently performs better than LAHC for almost all instances.

In this chapter, we have illustrated the success of multi-stage hyper-heuristics in combinatorial optimisation. We argue that the components of the developed multi-stage hyper-heuristic framework are reusable, and researchers can easily replace them with any methods of their choices. MSHH is an instance of the proposed framework, which provides the best general purpose multi-stage hyper-heuristic for heuristic search across different problem domains. MSHH is capable of generating new heuristics via relay hybridisation and then automatically identifying a useful subset of heuristics at a given

stage for local search. After associating each low level heuristic with a selection probability, stochastic local search starts with the incumbent set of low level heuristics using an adaptive move acceptance.

Chapter 6

Conclusion

6.1 Summary of Work

Search methodologies (i.e., heuristics) are at the core of almost all decision support systems, particularly while dealing with combinatorial optimisation problems. The state-of-the-art systems are often tailored for a particular problem by the experts in the area. Such systems are generally very costly to build and maintain. Since they are custom-made, it is almost impossible to apply/reuse them to/in another problem domain. Even a slight change in the problem definition could require an expert intervention. Whenever exact methods fail, researchers and practitioners resort to heuristics which are 'rule of thumb' methods for solving a given problem. There is a growing interest towards more general, cheaper and intelligent systems that can automate the heuristic design process. Humans design and provide the components of such systems while computers either run those components or use them to build new components while solving a given problem. Hyper-heuristics are such automated search methodologies that explore the space of heuristics for solving computationally difficult optimisation problems in decision support [2]. Hyper-heuristic research has been growing since the initial ideas have emerged in the 1960s [10, 11]. This thesis focuses on selection type hyper-heuristics, which were defined as 'heuristics to choose heuristics', initially [12]. An iterative selection hyper-heuristic passes a solution through a heuristic selection process to decide on a heuristic to apply from a fixed set of low level heuristics or move operators and then a move acceptance process to accept or reject the newly created solution at each step. The use of a logical interface between the high level hyper-heuristic and problem domain, referred to as *domain barrier* makes selection hyper-heuristics more general search methodologies than

the current techniques tailored for a particular domain. This barrier disallows a hyper-heuristic to retrieve any problem domain specific information. Hence, any selection hyper-heuristic (or its components) can be reused while solving any given problem, assuming that problem domain components have already been implemented. A goal in hyper-heuristic research is to raise the level of generality by providing automated hyper-heuristic solution methodologies that are able to self-tune/configure themselves and applicable to different problem domains without requiring any expert intervention and so additional development cost.

Selection hyper-heuristics are motivated by the reason that each heuristic performs differently on different instances and an approach mixing them could yield to a better overall performance. There is empirical evidence that the performance of selection hyper-heuristics could vary depending on the choice of heuristic selection and move acceptance components [8]. Hence, following the same argument, a simple framework which supports the design of easy-to-implement, easy-to-maintain and effective multi-stage hyper-heuristics allowing the use of multiple selection hyper-heuristics at different stages of the search process is proposed in this thesis. Six multi-stage hyper-heuristic methods are designed based on the framework and tested on a variety of problem domains.

A Dominance-based Random Descent/Gradient Hyper-heuristic with Naïve Move Acceptance (DRD) is proposed. The two heuristic selection methods are used in an alternating manner at successive stages. Greedy attempts to detect the low level heuristics with "good" performance and maintains a list of active heuristics considering the trade-off between the change (improvement) in the solution quality and the number of steps taken. If a heuristic takes a large number of successive steps and generating a large improvement in the solution quality, the performance of this heuristic is considered to be similar to the one which takes less number of successive steps and improves the solution quality less as well. Random descent selects from the (possibly) reduced set of low level heuristics to improve the solution in hand at each step. Whenever the search by random descent stagnates, then the greedy stage may restart for detecting new list of active heuristics.

Another effective multi-stage hyper-heuristic based on a round robin heuristic selection (Robinhood) (RHH) which allocates equal share from the overall time for each low level heuristic ordering them randomly within their categories of mutation and local search to process a solution in hand is proposed. The Robinhood hyper-heuristic operates in stages and prior to each stage, relevant decisions are made for the ordering of heuristics

within groups, parameters of the system components and the selection of one of the three move acceptance criteria.

A multi-stage hyper-heuristic, named Selection Hyper-heuristic with an Adaptive Threshold Acceptance (HySST), to intelligently and effectively exploit a suite of neighbourhood move operators is proposed. HySST is a stochastic search method which is significantly enhanced by a selection hyper-heuristic under a generalised iterated local search method. In HySST, two selection hyper-heuristics are employed operating cooperatively and mixing a set of domain-specific low level heuristics. The diversification-stage selection hyper-heuristic manages mutational move operators, while the intensification-stage selection hyper-heuristic mixes the hill climbing low level heuristics.

A multi-stage hyper-heuristic which combines two hyper-heuristics, Dominance-based hyper-heuristic and Roulette Wheel selection with Adaptive Threshold move acceptance (DRW) is proposed by extending DRD, RHH and HySST. A novel multi-stage hyper-heuristic approach which is based on the observation that not all low level heuristics for a problem domain would be useful at any point of the search process is proposed. The latter multi-stage hyper-heuristic named Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) extends the DRW approach and makes use of the *relay hybridisation* technique which applies a low level heuristic to a solution generated by applying a preceding heuristic.

A key goal in hyper-heuristic research is to build low cost methods which are general and can be reused on unseen problem instances as well as other problem domains desirably with no additional human expert intervention. Hence, the proposed multi-stage hyper-heuristic approaches are applied to six HyFlex problem domains to test the level of generality and some are further competed on several problem domains, mostly used in earlier international competitions leading to the main events ITC 2011, MISTA 2013, CHeSC 2011 and SolveIT 2011. Determining the state-of-the-art method among modern approaches for a given problem and providing a real world benchmark for comparison of approaches were the main deriving ideas behind the competitions.

The proposed multi-stage hyper-heuristics in this study are implemented as an extension to HyFlex (a software tool for hyper-heuristic development and research). The proposed learning Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance (MSHH) is deemed to be the winner. The results confirm its success when compared to each constituent hyper-heuristic, the

other multi-stage hyper-heuristics as well as the state-of-the-art hyper-heuristic which won the CHeSC 2011 competition. This multi-stage hyper-heuristic is a relatively simple approach which is easy-to-implement and easy-to-maintain compared to some previously proposed hyper-heuristics including the previous state-of-the-art hyper-heuristic, yet, it is extremely effective in cross-domain search delivering a superior performance.

High school timetabling problem is a real-world hard combinatorial optimisation problem. It seeks a search for the best event schedule and the best allocation of resources including the scheduling of classes, teachers, courses and students in time slots in a high school institution subject to a set of constraints. In a standard fashion, constraints are separated into *hard* and *soft*. The hard constraints must be satisfied in order to achieve *feasibility*, whereas the soft constraints characterise preferences and a solution for a given problem; solutions are expected to respect all hard constraints and satisfy as many soft constraints as possible. Hence, the violation of the soft constraint does not destroy the feasibility but rather affects the quality of the solution. In most of the previous formulations of the high school timetabling problem, infeasible solutions are allowed and evaluated, differentiating their quality by considering the degree of hard constraint violations. A unified high school timetabling problem which was a topic of a competition, referred to as ITC 2011, is described in this study. ITC 2011 provided a collection of high school timetabling problem instances collected from different countries across the world. The goal of the competition was to promote researchers and practitioners to deal with the real world complexities of the problem. As the team HySST (Hyper-heuristic Search Strategies and Timetabling), we joined the ITC 2011 with HySST multi-stage hyper-heuristic. Despite this being the first attempt in high school timetabling, the proposed approach of HySST generated the best new solutions for three given instances in Round 1 and gained the second place in Rounds 2 and 3.

A square matrix of distinct positive integers in which every row, column and diagonal has the same sum is called a magic square. The results show that the Dominance-based Roulette Wheel Multi-stage Hyper-heuristic using Relay Hybridisation and an Adaptive Threshold Acceptance is efficient enough in constructing the constrained-version of magic squares.

This study presents also the algorithm winning the MISTA 2013 Scheduling Challenge. The approach is a hybrid heuristic addressing an extension of the resource-constrained project scheduling problem. It comprises a Monte-Carlo tree search technique, very large scale neighbourhoods and applying the Dominance-based Roulette Wheel Hyper-heuristic with an Adaptive Threshold Acceptance (DRW) and highly optimised schedule

generator, all in the context of a multi-threaded population-based approach that uses ideas from memetic algorithms.

6.2 Discussion and Future Work

Overall success. The success of the proposed multi-stage hyper-heuristic approaches based on the proposed "simple" framework across a variety of domains provide empirical evidence that utilising and mixing the existing or new selection hyper-heuristics is indeed a good idea. The multi-stage hyper-heuristics designed based on the framework "(1) fast to implement, (2) requiring far less expertise in either the problem domain or heuristic methods, and (3) robust enough to effectively handle a range of problems". Moreover, the proposed multi-stage hyper-heuristic framework is general, reusable and useful in relieving the difficulty of choosing a hyper-heuristic method for solving a problem. Combining multiple hyper-heuristics always requires a decision on how long each stage should take and when to switch between selection hyper-heuristics at each stage. There are many ways to handle this, but simple choices seem to have performed really well. RHH handles both decisions in a static fixed way, while GGHH, DRD, HySST, DRW and MSHH handle both decisions adaptively, meaning that, if there is no improvement in consecutive stages while a certain hyper-heuristic is used then the other hyper-heuristic kicks in. In DRD and MSHH, this transition is probabilistic, while in GGHH, HySST and DRW another hyper-heuristic is invoked for certain. Adaptive approaches seem to perform better. MSHH is the current state-of-the-art multi-stage hyper-heuristic which has been tested on CHeSC 2011 and additional domains.

Reducing the set of low level heuristics. RHH does not reduce the set of low level heuristics and use all of them. HySST is based on an offline multi-stage hyper-heuristic managing a reduced set of low level heuristics, with the method employing either only mutational or only hill climbing low level heuristics at a given stage. The results revealed the success of HySST in solving the high school timetabling problem. On another hand, GGHH, DRD, DRW and MSHH methods dynamically reduce the set of heuristics using a greedy-like approach. This approach is a bi-objective learning approach considering time versus achieved solution quality trade-off, which discovers the most useful low level heuristics in improvement and at the same time generates their selection probabilities.

The results revealed the success of DRW in solving project scheduling problem, and MSHH in HyFlex. Yet, this approach has two main limitations: (i) it has a high run-time complexity, and (ii) the approach always favours the best low level heuristic which makes the largest improvement at any given step. This means that if the number of low level heuristics gets higher, the use of relay hybridisation could become impractical. In some problem domains, low level heuristics could have high running times due to their design. For example, in personnel scheduling of CHeSC 2011, some heuristics are very time consuming. Most of the algorithms are run as a contract algorithm and they have to terminate as soon as the time limit is exceeded. Running the greedy-like approach could use up most portion of that time limit whether relay hybridisation used or not and leaving less time for the algorithm to use what has been learnt, yielding to seemingly a "bad" performance of the algorithm. Considering a problem domain, each heuristic is generally designed with the goal of improving a given solution or get a given solution jump to the other potentially "good" regions of the search space. So, the question still remains whether it is always a good idea to focus solely on improvement and ignoring second best low level heuristics and others. It would be interesting to perform experiments on the problem domains incorporating a large number of low level heuristics (or time consuming low level heuristics) and develop a learning mechanism that overcomes the greedy-like weaknesses as future work.

Intensification and diversification. HySST and RHH use all provided low level heuristics for a given problem and require the heuristic type information, i.e., whether a low level heuristic is mutational or hill climbing to balance between diversification and intensification while selecting the low level heuristics. Iterated local search (ILS) enforces this balance by making use of two successive steps of perturbation and local search/hill climbing in its algorithmic framework (see [42] and Section 2.1.1). HySST and RHH also enforce the diversification and intensification in a different way. The main goal of a search method is always to make improvement as much as possible on the solution in hand. If the search stagnates (i.e. no improvement for a duration) then try to make a move even if worsening which will allow the method to explore different regions of the search space. HySST and RHH turns the whole framework into a random mutation hill climbing framework in which diversification is promoted first, but the relevant stages still attempt to make improvement using perturbative mutational heuristics. This stage is followed by local search using a set of pure hill climbing low level heuristics. In RHH, the transition is fixed, while HySST does that adaptively.

The other proposed multi-stage hyper-heuristic methods ignore the nature of the low

level heuristics. Considering the success of MSHH, enforcing greedy local search and so hill climbing for a short while for intensification purposes and then letting the algorithm to adaptively determine how to behave is the best strategy. In the latter stage, although the algorithm uses an adaptive threshold move acceptance, the overall algorithm does not always act as a diversifying component. Depending on the selected low level heuristic, the algorithm could behave as an intensification component. Because of this technique, a low level heuristic which combines a mutational and followed by a hill climbing heuristic (as observed in the experiments) just like in ILS could be chosen and applied. We argue that the incorporation of relay hybridisation technique within MSHH is one of the reasons making it the best multi-stage hyper-heuristic.

Adaptation of parameters. In a multi-stage hyper-heuristic, a low level heuristic, heuristic selection, move acceptance and multi-stage search control algorithm itself could introduce parameters. All those parameters should be either fixed via experimentation or the use of parameter tuning methods, such as F-race [38], REVAC [39], ParamILS [40] and/or preferably controlled using an appropriate method. Parameter values could be varied dynamically, changing their values in time using a predetermined strategy (such as in DRD) or adaptively, changing their values depending on the state of the search (such as in GGHH, RHH, HySST, DRW and MSHH). Although the multi-stage hyper-heuristic framework introduces another layer on multiple hyper-heuristics, this does not mean in any way that the resultant design will be any more complicated than the existing selection hyper-heuristics. Moreover, more parameters in the design of a hyper-heuristic could be a consequence of complexity of the search process using the provided components under a single control mechanism. Considering the success of MSHH, the use of the proposed multi-stage yielded a simple algorithm with less parameters (6 parameters) when compared to AdapHH [27] (the winning hyper-heuristic at CHeSC 2011) which introduced around 45 parameters[1]. This could be one of the reasons for the success of MSHH over AdapHH, because managing large number of parameters may distract the search process from focusing on finding good solutions and rather making the search focusing on adapting these parameters. It has been reported in [92] that the publicly available implementation of AdapHH counts over 3000 lines of code. The MSHH state-of-the-art multi-stage hyper-heuristic is around 10% the size of the code of the AdapHH implementation.

Crossover operators. There is still a debate going on the usefulness of crossover in evolutionary algorithms community [50, 51]. Considering that we have proposed

[1]This information is retrieved form the publicly available implementation of AdapHH

a single point based search framework, crossover operators provided in some of the problem domains are ignored by our multi-stage hyper-heuristics during the experiments, since crossover operators are mostly binary operators requiring two solutions as input necessitating another top level mechanism to decide on those solutions. Looking into how to best utilise all low level heuristics, including crossover operators within multi-stage hyper-heuristics would be of interest as future work.

Level of generality. Considering that the nature of problem instances could change in time, designing an algorithm which can handle those changes is crucial. This represents a level of generality as such an algorithm will be reusable even if the problem instances change in time. For example, [28] proposes a hyper-heuristic algorithm to handle the university timetabling problem. The hyper-heuristic design enables the algorithm to be reused and evaluated on other domains as well. So, the same hyper-heuristic is tested across CHeSC 2011 problem domains in [29].

Since hyper-heuristics separate adaptive search control from the details of the specific domain, it is naturally also envisaged the proposed multi-stage hyper-heuristics could be applied to other problems. In this thesis, we have tested some of our multi-stage hyper-heuristics on single domains initially, but then evaluated their level of generality on the six HyFlex problem domains. More importantly, their effectiveness and generality level can be further investigated on the future problem domains, that are implemented respecting the HyFlex interface, directly.

The ROADEF/EURO Challenge 2014 put forward a real-world problem of rolling stock unit management on railway sites subject to a range of constraints, including maintenance. The problem instances are provided by SNCF which is the national state-owned railway company of France. The goal of the competition is to determine the best approach which can handle multiple objectives. Rather than only using perturbative low level heuristics, we would like to study a multi-stage hyper-heuristic approach in order to exploit a set of constructive and perturbative low level heuristics, each of which attempts to enhance an aspect of the quality of a solution in hand during the search process. Although the proposed multi-stage hyper-heuristic framework is used with a set of perturbative low level heuristics in this thesis, it enables the use of constructive and perturbative low level heuristics separating both processes into stages. An initial solver based on the development of multi-stage hyper-heuristics was developed to produce a feasible solution with fairly good quality, and obtained the 3rd prize in the Junior Category of the ROADEF/EURO Challenge 2014.

Bibliography

[1] D. L. Anderson. Magic squares. `http://illuminations.nctm.org/Lesson.aspx?id=655`, 2001.

[2] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[3] E. K. Burke, 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*, pages 457–474. Kluwer, 2003.

[4] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. A classification of hyper-heuristic approaches. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research and Management Science*, pages 449–468. Springer US, 2010.

[5] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In Janusz Kacprzyk, Lakhmi C. Jain, Christine L. Mumford, and Lakhmi C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009.

[6] P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. Springer, 2005.

[7] Edmund Burke, Graham Kendall, Mustafa Misir, and Ender Özcan. Monte Carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196 (1):73–90, 2012.

[8] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[9] Ender Özcan and Andrew J. Parkes. Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 2011–2018, New York, NY, USA, 2011. ACM.

[10] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251, New Jersey, 1963. Prentice-Hall, Inc.

[11] W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. *Probabilistic and parametric learning combinations of local job shop scheduling rules*. 117. Defense Technical Information Center, 1963.

[12] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin Heidelberg, 2001.

[13] E. Özcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In Thomas Philip Runarsson, Hans-Georg Beyer, Edmund Burke, Juan J. Merelo-Guervós, L. Darrell Whitley, and Xin Yao, editors, *Parallel Problem Solving from Nature (PPSN IX)*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211. Springer Berlin Heidelberg, 2006.

[14] G. Kendall and M. Mohamad. Channel assignment optimisation using a hyper-heuristic. In *Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent Systems (CIS2004)*, pages 790–795, Singapore, 2004.

[15] M. Ayob and G. Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies (InTech'03)*, pages 132–141, Chiang Mai, Thailand, 2003.

[16] E. Özcan, Y. Bykov, M. Birben, and E. K. Burke. Examination timetabling using late acceptance hyper-heuristics. In *IEEE Congress on Evolutionary Computation (CEC '09)*, pages 997–1004, 2009.

[17] E. K. Burke, J. D. Landa-Silva, and E. Soubeiga. Multi-objective hyper-heuristic approaches for space allocation and timetabling. In Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces Series*, pages 129–158. Springer US, 2005.

[18] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[19] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende and J. P. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, chapter 9, pages 523–544. Kluwer, 2003.

[20] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403–2435, 2007.

[21] K. A. Dowsland, E. Soubeiga, and E. K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.

[22] D. Ouelhadj and S. Petrovic. A cooperative distributed hyper-heuristic framework for scheduling. In *Proceedings of the IEEE International Conference on Man, Cybernetics, and Systems*, pages 1232–1238, 2008.

[23] B. Bilgin, E. Özcan, and E. E. Korkmaz. An experimental study on hyper-heuristics and exam scheduling. In Edmund K. Burke and Hana Rudová, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 394–412. Springer Berlin Heidelberg, 2007.

[24] Per Kristian Lehre and Ender Özcan. A runtime analysis of simple hyper-heuristics: to mix or not to mix operators. In *Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms (FOGA XII '13)*, pages 97–104, New York, NY, USA, 2013. ACM.

[25] Ender Özcan, Mustafa Misir, and Ahmed Kheiri. Group decision making hyper-heuristics for function optimisation. In *13th UK Workshop on Computational Intelligence (UKCI2013)*, pages 327–333. IEEE, 2013.

[26] E. Özcan and E. K. Burke. Multilevel search for choosing hyper-heuristics. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Application (MISTA2009)*, pages 788–789, 2009.

[27] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. A new hyper-heuristic implementation in HyFlex: a study on generality. In John Fowler, Graham Kendall, and Barry McCollum, editors, *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory and Application (MISTA2011)*, pages 374–393, 2011.

[28] Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In *12th UK Workshop on Computational Intelligence (UKCI2012)*, pages 1–8. IEEE, 2012.

[29] Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing*, 17(12): 2279–2292, 2013.

[30] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[31] Maurice Kraitchik. Magic squares. In *Mathematical Recreations*, chapter 7, pages 142–192. New York: Norton, 1942.

[32] Tao Xie and Lishan Kang. An evolutionary algorithm for magic squares. In *IEEE Congress on Evolutionary Computation (CEC '03)*, volume 2, pages 906–913, 2003.

[33] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, and E. K. Burke. HyFlex: a benchmark framework for cross-domain heuristic search. In J.-K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 136–147. Springer Berlin Heidelberg, 2012.

[34] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[35] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[36] Kenneth Sörensen and Fred W. Glover. Metaheuristics. In Saul I. Gass and Michael C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer US, 2013.

[37] Kumara Sastry, David E. Goldberg, and Graham Kendall. Genetic algorithms. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 93–117. Springer US, 2014.

[38] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[39] Volker Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 975–980, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[40] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[41] E. H. L. Aarts and J. K. Lenstra. *Local search in combinatorial optimization.* Princeton University Press, 2003.

[42] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search: framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research and Management Science*, pages 363–397. Springer US, 2010.

[43] Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. A stochastic local search algorithm with adaptive acceptance for high-school timetabling. *Annals of Operations Research*, to appear.

[44] D. A. Abramson, H. Dang, and M. Krisnamoorthy. Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, 16(1):1–22, 1999.

[45] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[46] Jonathan M. Thompson and Kathryn A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25 (7-8):637–648, 1998.

[47] Gunter Dueck. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993.

[48] Edmund K. Burke and Yuri Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT '08)*, 2008.

[49] Warren G. Jackson, Ender Özcan, and John H. Drake. Late acceptance-based selection hyper-heuristics for cross-domain heuristic search. In *13th UK Workshop on Computational Intelligence (UKCI2013)*, pages 228–235. IEEE, 2013.

[50] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 539–546, New York, NY, USA, 2008. ACM.

[51] Melanie Mitchell, John H. Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 51–58. Morgan Kaufmann, 1994.

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

[53] Pablo Moscato and Michael G. Norman. A "memetic" approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In *In Proceedings of the International Conference on Parallel Computing and Transputer Applications*, pages 177–186. IOS Press, 1992.

[54] Ferrante Neri and Carlos Cotta. Memetic algorithms and memeting computing optimization: a literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[55] Daniel Karapetyan and Gregory Gutin. A new approach to population sizing for memetic algorithms: a case study for the multidimensional assignment problem. *Evolutionary Computation*, 19(3):345–371, 2011.

[56] Ender Özcan, Andrew J. Parkes, and Alpay Alkan. The interleaved constructive memetic algorithm and its application to timetabling. *Computers and Operations Research*, 39(10):2310–2322, 2012.

[57] Gregory Gutin and Daniel Karapetyan. A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60, 2009.

[58] Ender Özcan. Memes, self-generation and nurse rostering. In Edmund K. Burke and Hana Rudova, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 85–104. Springer Berlin Heidelberg, 2007.

[59] P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *IEEE Congress on Evolutionary Computation (CEC '03)*, pages 1214–1221, 2003.

[60] Edmund K. Burke, Timothy Curtois, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, José Antonio Vázquez Rodríguez, and Michel Gendreau. Iterated local search vs. hyper-heuristics: towards general-purpose search algorithms. In *IEEE Congress on Evolutionary Computation (CEC '10)*, pages 1–8, 2010.

[61] Mustafa Misir, Wim Vancroonenburg, Katja Verbeeck, and Greet Vanden Berghe. A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In Luca Di Gaspero, Andrea Schaerf, and Thomas Stützle, editors, *Proceedings of the 9th Metaheuristics International Conference (MIC2011)*, pages 289–298, 2011.

[62] B. Kiraz, A. S. Uyar, and E. Özcan. Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769, 2013.

[63] Haluk Rahmi Topcuoglu, Abdulvahid Ucar, and Lokman Altin. A hyper-heuristic based framework for dynamic optimization problems. *Applied Soft Computing*, 19: 236–251, 2014.

[64] Kent McClymont and Edward C. Keedwell. Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 2003–2010, New York, NY, USA, 2011. ACM.

[65] Mashael Maashi, Ender Özcan, and Graham Kendall. A multi-objective hyper-heuristic based on choice function. *Expert Systems with Applications*, 41(9):4475–4493, 2014.

[66] N. Pillay. A study of hyper-heuristics for hybridizing search. In *Advances in Artificial Intelligence - Proceedings of ALEA*, pages 128–139, 2013.

[67] Jerry Swan, John Woodward, Ender Özcan, Graham Kendall, and Edmund Burke. Searching the hyper-heuristic design space. *Cognitive Computation*, 6(1):66–73, 2014.

[68] Djamila Ouelhadj, Sanja Petrovic, and Ender Özcan. A multi-level search framework for asynchronous cooperation of multiple hyper-heuristics. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*, pages 2193–2196, New York, NY, USA, 2009. ACM.

[69] Konstantin Chakhlevitch and Peter Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In Günther Raidl and Jens Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *Lecture Notes in Computer Science*, pages 23–33. Springer Berlin Heidelberg, 2005.

[70] Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: recent developments. In Carlos Cotta, Marc Sevaux, and Kenneth Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer Berlin Heidelberg, 2008.

[71] Jerry Swan, Ender Özcan, and Graham Kendall. Hyperion - a recursive hyper-heuristic framework. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 616–630. Springer Berlin Heidelberg, 2011.

[72] José Luis Núñez and Alberto Ceballos. A general purpose hyper-heuristic based on ant colony optimization. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`, 2011.

[73] Ping-Che Hsiao, Tsung-Che Chiang, and Li-Chen Fu. A VNS-based hyper-heuristic with adaptive computational budget of local search. In *IEEE Congress on Evolutionary Computation (CEC '12)*, pages 1–8, 2012.

[74] Tomasz Cichowicz, Maciej Drozdowski, Michal Frankiewicz, Grzegorz Pawlak, Filip Rytwinski, and Jacek Wasilewski. Five phase and genetic hive hyper-heuristics for the cross-domain search. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 354–359. Springer Berlin Heidelberg, 2012.

[75] Mathieu Larose. A hyper-heuristic for the CHeSC 2011. In *The 53rd Annual Conference of the UK Operational Research Society (OR53)*, 2011.

[76] Mark Johnston, Thomas Liddle, Joel Miller, and Mengjie Zhang. A hyperheuristic based on dynamic iterated local search. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`, 2011.

[77] C. Y. Chan, Fan Xue, W. H. Ip, and C. F. Cheung. A hyper-heuristic inspired by pearl hunting. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 349–353. Springer Berlin Heidelberg, 2012.

[78] David Meignan. An evolutionary programming hyper-heuristic with co-evolution for CHeSCÍ1. In *The 53rd Annual Conference of the UK Operational Research Society (OR53)*, 2011.

[79] Andreas Lehrbaum and Nysret Musliu. A new hyperheuristic algorithm for cross-domain search problems. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 437–442. Springer Berlin Heidelberg, 2012.

[80] Luca Di Gaspero and Tommaso Urli. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 384–389. Springer Berlin Heidelberg, 2012.

[81] Franco Mascia and Thomas Stützle. A non-adaptive stochastic local search algorithm for the CHeSC 2011 competition. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 101–114. Springer Berlin Heidelberg, 2012.

[82] Alberto Acuña, Victor Parada, and Gustavo Gatica. Cross-domain heuristic search challenge: GISS algorithm presentation. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`, 2011.

[83] Jiří Kubalík. Hyper-heuristic based on iterated local search driven by evolutionary algorithm. In J.-K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 148–159. Springer Berlin Heidelberg, 2012.

[84] Jawad Elomari. Self-search. `http://www.asap.cs.nott.ac.uk/external/chesc2011/`, 2011.

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

[86] Kent McClymont and Edward Keedwell. A single objective variant of the online selective Markov chain hyper-heuristic (MCHH-S). `http://www.asap.cs.nott.ac.uk/external/chesc2011/`, 2011.

[87] Jonatan Gomez. Self adaptation of operator rates in evolutionary algorithms. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation (GECCO '04)*, volume 3102 of *Lecture Notes in Computer Science*, pages 1162–1173. Springer Berlin Heidelberg, 2004.

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

[89] John H. Drake, Ender Özcan, and Edmund K. Burke. An improved choice function heuristic selection for cross domain heuristic search. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving From Nature (PPSN XII)*, volume 7492 of *Lecture Notes in Computer Science*, pages 307–316. Springer Berlin Heidelberg, 2012.

[90] Edmund K. Burke, Michel Gendreau, Gabriela Ochoa, and James D. Walker. Adaptive iterated local search for cross-domain optimisation. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 1987–1994, New York, NY, USA, 2011. ACM.

[91] Gabriela Ochoa, James Walker, Matthew Hyde, and Tim Curtois. Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic framework. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature (PPSN XII)*, volume 7492 of *Lecture Notes in Computer Science*, pages 418–427. Springer Berlin Heidelberg, 2012.

[92] Steven Adriaensen, Tim Brys, and Ann Nowé. Fair-share ILS: a simple state-of-the-art iterated local search hyperheuristic. In Dirk V. Arnold, editor, *Proceedings*

of the 2014 Conference on Genetic and Evolutionary Computation (GECCO '14), pages 1303–1310, New York, NY, USA, 2014. ACM.

[93] Sol Broder. Final examination scheduling. *Communications of the ACM*, 7(8): 494–498, 1964.

[94] D. de Werra. The combinatorics of timetabling. *European Journal of Operational Research*, 96(3):504–513, 1997.

[95] Nelishia Pillay. An overview of school timetabling research. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT '10)*, pages 321–335, 2010.

[96] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2007.

[97] Rhydian Lewis, Ben Paechter, and Olivia Rossi-Doria. Metaheuristics for university course timetabling. In Keshav P. Dahal, Kay Chen Tan, and Peter I. Cowling, editors, *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*, pages 237–272. Springer Berlin Heidelberg, 2007.

[98] Wilhelm Erben and Jürgen Keppler. A genetic algorithm solving a weekly course-timetabling problem. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling I*, volume 1153 of *Lecture Notes in Computer Science*, pages 198–211. Springer Berlin Heidelberg, 1996.

[99] Krzysztof Socha, Joshua Knowles, and Michael Sampels. A MAX-MIN ant system for the university course timetabling problem. In Marco Dorigo, Gianni Caro, and Michael Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 2002.

[100] Peter Ross, Dave Corne, and Hsiao-Lan Fang. Improving evolutionary timetabling with delta evaluation and directed mutation. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature (PPSN III)*, volume 866 of *Lecture Notes in Computer Science*, pages 556–565. Springer Berlin Heidelberg, 1994.

[101] Dave Corne, Peter Ross, and Hsiao-Lan Fang. Fast practical evolutionary timetabling. In Terence C. Fogarty, editor, *Evolutionary Computing*, volume 865 of *Lecture Notes in Computer Science*, pages 250–263. Springer Berlin Heidelberg, 1994.

[102] D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, 37(1):98–113, 1991.

[103] A. Colorni, M. Dorigo, and V. Maniezzo. A genetic algorithm to solve the timetable problem. Technical Report Technical Report No. 90-060, Politecnico di Milano, Italy, 1992.

[104] Alain Hertz. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35(3):255–270, 1992.

[105] Khang Nguyen Tan Tran Minh, Nguyen Dang Thi Thanh, Khon Trieu Trang, and Nuong Tran Thi Hue. Using tabu search for solving a high school timetabling problem. In Ngoc Thanh Nguyen, Radoslaw Katarzyniak, and Shyi-Ming Chen, editors, *Advances in Intelligent Information and Database Systems*, volume 283 of *Studies in Computational Intelligence*, pages 305–313. Springer Berlin Heidelberg, 2010.

[106] G. S. Bello, M. C. Rangel, and M. C. S. Boeres. An approach for the class/teacher timetabling problem. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT '08)*, 2008.

[107] Andrea Schaerf. Tabu search techniques for large high-school timetabling problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 363–368. AAAI Press, 1996.

[108] Grigorios N. Beligiannis, Charalampos N. Moschopoulos, Georgios P. Kaperonis, and Spiridon D. Likothanassis. Applying evolutionary computation to the school timetabling problem: the Greek case. *Computers and Operations Research*, 35(4): 1265–1280, 2008.

[109] Frank Jacobsen, Andreas Bortfeldt, and Hermann Gehring. Timetabling at German secondary schools: tabu search versus constraint programming. In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT '06)*, pages 439–442, 2006.

[110] Geraldo Ribeiro Filho, Luiz Antonio, and Luiz Antonio Nogueira Lorena. A constructive evolutionary approach to school timetabling. In Egbert J. W. Boers, editor, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 130–139. Springer Berlin Heidelberg, 2001.

[111] Peter Wilke, Matthias Gröbner, and Norbert Oster. A hybrid genetic algorithm for school timetabling. In Bob McKay and John Slaney, editors, *AI 2002: Advances in Artificial Intelligence*, volume 2557 of *Lecture Notes in Computer Science*, pages 455–464. Springer Berlin Heidelberg, London, UK, 2002.

[112] Rushil Raghavjee and Nelishia Pillay. An application of genetic algorithms to the school timetabling problem. In *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology (SAICSIT '08)*, pages 193–199, New York, NY, USA, 2008. ACM.

[113] D. Abramson and H. Dang. School timetables: a case study in simulated annealing. In René V. V. Vidal, editor, *Applied Simulated Annealing*, volume 396 of *Lecture Notes in Economics and Mathematical Systems*, pages 103–124. Springer Berlin Heidelberg, 1993.

[114] Rushil Raghavjee and Nelishia Pillay. Using genetic algorithms to solve the South African school timetabling problem. In *Second World Congress on Nature and Biologically Inspired Computing (NaBIC 2010)*, pages 286–292. IEEE, 2010.

[115] Anjuli Kannan, Gerald van den Berg, and Adeline Kuo. ischedule to personalize learning. *Interfaces*, 42(5):437–448, 2012.

[116] A. Alkan and E. Özcan. Memetic algorithms for timetabling. In *IEEE Congress on Evolutionary Computation (CEC '03)*, volume 3, pages 1796–1802, 2003.

[117] Arnaldo Vieira Moura and Rafael Augusto Scaraficci. A GRASP strategy for a more constrained school timetabling problem. *International Journal of Operational Research*, 7(2):152–170, 2010.

[118] Nelishia Pillay. A study into the use of hyper-heuristics to solve the school timetabling problem. In *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT '10)*, pages 258–264, New York, NY, USA, 2010. ACM.

[119] Nelishia Pillay. Hyper-heuristics for educational timetabling. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT '12)*, 2012.

[120] Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the

research agenda in automated timetabling: the second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.

[121] G. Post. Benchmarking project for high school timetabling. `http://www.utwente.nl/ctit/hstt/`, 2011.

[122] Gerhard Post, Luca Di Gaspero, Jeffrey H. Kingston, Barry McCollum, and Andrea Schaerf. The third international timetabling competition. *Annals of Operations Research*, pages 1–7, 2013.

[123] Jonathan Domrös and Jörg Homberger. An evolutionary algorithm for high school timetabling. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT '12)*, pages 485–488, 2012.

[124] Matias Sørensen, Simon Kristiansen, and Thomas R. Stidsen. International timetabling competition 2011: an adaptive large neighborhood search algorithm. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT '12)*, pages 489–492, 2012.

[125] George H. G. Fonseca, Samuel S. Brito, and Haroldo G. Santos. A simulated annealing based approach to the high school timetabling problem. In Hujun Yin, José A. F. Costa, and Guilherme Barreto, editors, *Intelligent Data Engineering and Automated Learning (IDEAL 2012)*, volume 7435 of *Lecture Notes in Computer Science*, pages 540–549. Springer Berlin Heidelberg, 2012.

[126] Nelishia Pillay. A survey of school timetabling research. *Annals of Operations Research*, pages 1–33, 2013.

[127] I. Kurtulus and E. W. Davis. Multi-project scheduling: categorization of heuristic rules performance. *Management Science*, 28(2):161–172, 1982.

[128] Peter Brucker, Andreas Drexl, Rolf Mohring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[129] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers and Operations Research*, 25(4):279–302, 1998.

[130] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.

[131] Linet Özdamar and Gündüz Ulusoy. A survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27(5):574–586, 1995.

[132] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[133] Jan Weglarz, Joanna Józefowska, Marek Mika, and Grzegorz Waligóra. Project scheduling with finite or infinite number of activity processing modes - a survey. *European Journal of Operational Research*, 208(3):177–205, 2011.

[134] Amol Singh. Resource constrained multi-project scheduling with priority rules & analytic hierarchy process. *Procedia Engineering*, 69:725–734, 2014.

[135] Ching-Chih Tseng. Two heuristic algorithms for a multi-mode resource-constrained multi-project scheduling problem. *Journal of Science and Engineering Technology*, 4(2):63–74, 2008.

[136] S. E. Elmaghraby. *Activity networks: project planning and control by network models*. Wiley-Interscience Publication. Wiley, 1977.

[137] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[138] Roman Słowinski. Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. *European Journal of Operational Research*, 7(3): 265–273, 1981.

[139] R. P. Mohanthy and M. K. Siddiq. Multiple projects-multiple resources-constrained scheduling: some studies. *International Journal of Production Research*, 27(2):261–280, 1989.

[140] Richard F. Deckro, E.P. Winkofsky, John E. Hebert, and Roger Gagnon. A decomposition approach to multi-project scheduling. *European Journal of Operational Research*, 51(1):110–118, 1991.

[141] A. Alan B. Pritsker, Lawrence J. Watters, and Philip M. Wolfe. Multiproject scheduling with limited resources: a zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

[142] Kwan Woo Kim, Young Su Yun, Jung Mo Yoon, Mitsuo Gen, and Genji Yamazaki. Hybrid genetic algorithm with adaptive abilities for resource-constrained multiple project scheduling. *Computers in Industry*, 56(2):143–160, 2005. Applications of Genetic Algorithms in Industry.

[143] S. Kumanan, G. Jegan Jose, and K. Raja. Multi-project scheduling using an heuristic and a genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 31(3-4):360–366, 2006.

[144] J. F. Gonçalves, J. J. M. Mendes, and M. G. C. Resende. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190, 2008.

[145] John H Payne. Management of multiple simultaneous projects: a state-of-the-art review. *International Journal of Project Management*, 13(3):163–168, 1995.

[146] Matthew J. Liberatore and George J. Titus. The practice of management science in R&D project management. *Management Science*, 29(8):962–974, 1983.

[147] M. Liberatore, B. Pollack-Johnson, and C. Smith. Project management in construction: software use and research directions. *Journal of Construction Engineering and Management*, 127(2):101–107, 2001.

[148] Anıl Can and Gündüz Ulusoy. Multi-project scheduling with two-stage decomposition. *Annals of Operations Research*, 217(1):95–116, 2014.

[149] Jiuping Xu and Cuiying Feng. Multimode resource-constrained multiple project scheduling problem under fuzzy random environment and its application to a large scale hydropower construction project. *The Scientific World Journal*, 2014: published online `http://dx.doi.org/10.1155/2014/463692`, 2014.

[150] Chunhua Ju and Tinggui Chen. Simplifying multiproject scheduling problem based on design structure matrix and its solution by an improved aiNet algorithm. *Discrete Dynamics in Nature and Society*, 2012:published online `http://dx.doi.org/10.1155/2012/713740`, 2012.

[151] Rainer Kolisch and Arno Sprecher. PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997.

[152] Martin Josef Geiger. Iterated variable neighborhood search for the resource constrained multi-mode multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 807–811, Ghent, Belgium, 2013.

[153] T. A. M. Toffolo, H. G. Santos, M. Carvalho, and J. Soares. An integer programming approach for the multi-mode resource-constrained multiproject scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 840–847, Ghent, Belgium, 2013.

[154] Haroldo G. Santos, Janniele Soares, and T. Toffolo. Hybrid local search for the multi-mode resource-constrained multi-project scheduling problem. In *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling (PATAT '14)*, pages 397–407, 2014.

[155] Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 836–839, Ghent, Belgium, 2013.

[156] C. Artigues and E. Hebrard. MIP relaxation and large neighborhood search for a multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 815–819, Ghent, Belgium, 2013.

[157] L. M. Borba, A. J. Benavides, T. Zubaran, G. C. Carniel, and M. Ritt. A simple stochastic local search for multi-mode resource-constrained multi-project scheduling. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 826–830, Ghent, Belgium, 2013.

[158] F. Alonso-Pecina, J. E. Pecero, and D. Romero. A three-phases based algorithm for the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th*

Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013), pages 812–814, Ghent, Belgium, 2013.

[159] M. López-Ibáñez, F. Mascia, M. E. Marmion, and T. Stützle. Automatic design of a hybrid iterated local search for the multi-mode resource constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 820–825, Ghent, Belgium, 2013.

[160] H. Bouly, D. C. Dang, A. Moukrim, and H. Xu. Evolutionary algorithm and post-processing to minimize the total delay in multi-project scheduling. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 831–835, Ghent, Belgium, 2013.

[161] A. Schnell. A hybrid local search algorithm integrating an exact cp-sat approach to solve the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013)*, pages 802–806, Ghent, Belgium, 2013.

[162] Eric W. Weisstein. Magic squares. `http://mathworld.wolfram.com/MagicSquare.html`, 2003.

[163] Gakuho Abe. Unsolved problems on magic squares. *Discrete Mathematics*, 127 (1-3):3–13, 1994.

[164] K. Pinn and C. Wieczerkowski. Number of magic squares from parallel tempering Monte Carlo. *International Journal of Modern Physics C*, 9(4):541–546, 1998.

[165] Edmund K. Burke and Yuri Bykov. The late acceptance hill-climbing heuristic. Technical Report Technical Report No. CSM-192, Computing Science and Mathematics, University of Stirling, 2012.

[166] Ahmed Kheiri and Ender Özcan. Constructing constrained-version of magic squares using selection hyper-heuristics. *The Computer Journal*, 57(3):469–479, 2014.

[167] Gerhard Post, Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngas, Cimmo Nurmi, and David Ranson. An XML format for benchmarks in high school timetabling. *Annals of Operations Research*, 194(1):385–397, 2012.

[168] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[169] Yew-Soon Ong, Meng-Hiot Lim, Ning Zhu, and Kok-Wai Wong. Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):141–152, 2006.

[170] Ender Özcan and Ahmed Kheiri. A hyper-heuristic based on random gradient, greedy and dominance. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Computer and Information Sciences II*, pages 557–563. Springer London, 2012.

[171] Ahmed Kheiri and Ender Özcan. A hyper-heuristic with a round robin neighbourhood selection. In Martin Middendorf and Christian Blum, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7832 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2013.

[172] Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. HySST: hyper-heuristic search strategies and timetabling. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT '12)*, pages 497–499, 2012.

[173] Shahriar Asta, Daniel Karapetyan, Ahmed Kheiri, Ender Özcan, and Andrew J. Parkes. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Special Issue of Journal of Scheduling*, under review.

[174] Ahmed Kheiri and Ender Özcan. A multi-stage selection hyper-heuristic. *EJOR*, under review.

[175] E. Özcan and E. Ersoy. Final exam scheduler - FES. In *IEEE Congress on Evolutionary Computation (CEC '05)*, volume 2, pages 1356–1363, 2005.

[176] R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. McCollum. Memory length in hyper-heuristics: an empirical study. In *IEEE Symposium on Computational Intelligence in Scheduling (SCIS '07)*, pages 173–178, 2007.

[177] R. Bai, E. K. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. Technical Report Technical Report NOTTCS-TR-2007-8, School of CSiT, University of Nottingham, 2007.

[178] Kalyanmoy Deb. Multi-objective genetic algorithms: problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.

[179] Henry Kautz, Eric Horvitz, Yongshao Ruan, Carla Gomes, and Bart Selman. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial intelligence (AAAI'02)*, pages 674–681, Menlo Park, CA, USA, 2002. AAAI Press.

[180] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[181] William H. Kruskal. Historical notes on the Wilcoxon unpaired two-sample test. *Journal of the American Statistical Association*, 52(279):356–360, 1957.

[182] Morten W. Fagerland and Leiv Sandvik. The Wilcoxon–Mann–Whitney test under scrutiny. *Statistics in Medicine*, 28(10):1487–1497, 2009.