SEARCH AND SELECTION METHODS FOR HYPER-HEURISTICS

BY

STEPHEN M. AKANDWANAHO

SUBMITTED IN PARTIAL FULFILLMENT OF THE ACADEMIC REQUIREMENTS OF DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

UNIVERSITY OF KWAZULU-NATAL

COLLEGE OF AGRICULTURE, ENGINEERING AND SCIENCE

SCHOOL OF COMPUTER SCIENCE

WESTVILLE CAMPUS

MAY, 2018

Declaration

The research work described in this thesis was carried out in the School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal, Westville campus, under the supervision of Professor Serestina Viriri.

I, Stephen Mugisha Akandwanaho, declare that this thesis is my own, unaided work. It has not been submitted in any form for any degree or diploma to any other tertiary institution. Where use has been made of the work of others, it is duly acknowledged in the text.

As a candidate's supervisor, I, Professor Serestina Viriri, have approved this dissertation for submission.

Signed: Stephen M. Akandwanaho
Date: 22nd May 2018

Signed: Professor Serestina Viriri
Supervisor (on behalf of the HDR)
Date: 22nd May 2018

Publications

1. S.M. Akandwanaho, A.O. Adewumi, and A.A. Adebiyi, "Solving Dynamic Traveling Salesman Problem Using Dynamic Gaussian Process Regression" Journal of Applied Mathematics, vol. 2014, Article ID 818529, 10 pages, 2014. doi:10.1155/2014/818529.

2. S.M. Akandwanaho and A.O. Adewumi, "Stochastic Optimization for Renewable Energy Grid Sharing in Sub-Saharan Africa Using Memetic Algorithm" Proceedings of the World Engineering Conference on Sustainable Infrastructure, 1-13, WECSI 2014.

3. S.M. Akandwanaho and S.Viriri, "A Spy Search Mechanism (SSM) for Memetic Algorithm (MA) in dynamic environments"International Workshop on Multi-disciplinary Trends in Artificial Intelligence LNCS - Springer, 450-461, 2017.

4. S.M. Akandwanaho and S.Viriri, "An Efficient Choice Function (ECF) for Hyper-Heuristics" Submitted to Journal of the Operational Research Society (JORS).

5. S.M. Akandwanaho and S.Viriri, "Predictive Memetic Algorithm (PMA) for Combinatorial Optimization in Dynamic Environments" Submitted to an international scientific conference for research in the field of Computational Collective Intelligence, Bristol, UK.

Abstract

Hyper-heuristics (HHs) search from a search space of heuristics for an optimal heuristic that can be mapped onto a problem to generate good solutions. One of the heuristic selection methods used by hyper-heuristics is a choice function (CF) which assigns scores to heuristics according to their performance. An investigation is conducted on a choice function for single-point selective hyper-heuristics. The drawbacks of the existing choice function are: discarding heuristics due to poor performance on a problem when they could exhibit good performance on another problem and premature convergence in the heuristic search space. In order to address the drawbacks of a choice function, a new selection method called an efficient choice function (ECF) is introduced, based on a three-pronged improvement approach.

Firstly, a new element is introduced, which collects previously poor-performing heuristics. The best heuristic of the poorly performing heuristics is obtained and compared with the best heuristic from the general pool of heuristics. Secondly, the pairwise comparison of the best heuristics from both the poorly performing heuristics and the general pool of heuristics is applied at every point in the iteration to maintain competition throughout the search process and generate high-quality outcomes. Thirdly, another element is introduced that randomly divides heuristics into different groups, ranks the collective performance of each group and makes performance comparisons between disparate groups of heuristics. The proposed heuristic selection method is tested on several well-known combinatorial optimization problems which include, the vehicle routing problem, the bin packing problem, the permutation flow shop problem, the personnel scheduling problem and the patient transportation problem. Results show a better performance of an efficient choice function than the existing methods.

The second contribution of this thesis is to enhance searching for optima in dynamic search spaces. An investigation of dynamic selection hyper-heuristics is performed and a new non-stationary covariance function is introduced. The Gaussian process regression is used as a predictive measure for moving optima in dynamic search environments and the proposed method is applied to the dynamic traveling salesman problem, which yields better performance than the existing approach.

The third contribution is a spy search method (SSM) for a memetic algorithm (MA) in dynamic environments. Given that the proposed efficient choice function for hyper-heuristics is based on a memetic to perform the search for an optimal heuristic, improving a MA search enhances the capacity of the efficient choice function to find good solutions. The proposed SSM shows a better performance than the nine existing methods on a set of different dynamic problems.

Acknowledgments

The entire work in this thesis was supervised by Professor Serestina Viriri. I thank him for guiding me in making this thesis possible. Without him this work would not have been possible. Thank you Professor Viriri.

I thank my family and friends, especially my wife, for standing with me throughout my studies.

Finally, I thank my Lord God Almighty who began this work in me and continued it to the end (Philippians 1:6).

Contents

List of Figures

List of Tables

Acronyms

AHC adaptive hill climbing

AHMA memetic algorithm with adaptive hill climbing operator

CF choice function

CPSO-MHS cooperative particle swarm optimizer modified harmony search

ECF efficient choice function

DAR dial-a-ride

DMA dynamic memetic algorithm

DOOs dynamic optimization problems

DPGR dynamic Gaussian process regression

DTP dynamic transportation problem

DTSP dynamic transport salesman problem

GA genetic algorithm

GP genetic programming

GPR Gaussian process regression

GD great deluge

GCHC greedy crossover hill climbing

HC hill climbing

HH hyper-heuristics

HSA-M harmony search algorithm with memory mechanism

HyFlex hyper-heuristics flexible framework

ILS iterated local search

LLHs low level heuristics

LS local search

MA memetic algorithm

mQSO multi swarm quantum particle swarm optimization

NASA national aeronautics and space administration

NP hard non-deterministic polynomial-time hard

PSO particle swarm optimization

PTP patient transportation problem

RIGA genetic algorithm with random immigrants scheme

SA simulated annealing

SSM spy-search mechanism

TSP transport salesman problem

VNS variable neighborhood search

VRP vehicle routing problem

VRPTW vehicle routing problem with time windows

Chapter 1

Introduction

Hyper-heuristics (HHs) are relatively new methods compared to other optimization methods. They were born out of the traditional meta-heuristics, which are inherently problem dependent techniques. HHs are problem independent, since they produce generic methods that are applied to various problems. These methods search within a pool of problem-independent heuristics. Meta-heuristics, on the other hand, produce tailored methods which are problem dependent [1]. Hyper-heuristics operate at the abstraction layer. This layer is where generic information about problem domains is accepted through the domain barrier in the hyper-heuristics structure. In order to select a heuristic to map onto a problem, a choice function (CF) search method is used. This is a key method that HHs use to search for a desired one from the sample of heuristics [2]. The performance of HHs depends on the efficiency of a choice function. For instance, a weak heuristic selected by a CF cannot yield desired outcomes when mapped onto a problem. Moreover, many search spaces are of a dynamic nature. This increases the complexity of problems, hence a demand for dynamic methods that can adequately explore the search for high quality solutions. However, the existing CF presents several shortcomings [3–6] which include:

1. Discarding a heuristic due to its dismal performance on a problem. A heuristic may yield good outcomes when mapped onto a different problem. There is no mechanism in selective hyper-heuristics for storing and re-calling poorly performing heuristics in order to map them onto suitable problems instead of generating new heuristics.

2. Inaccuracy of the scoring process which is based on the synergy between paired heuristics. Heuristics are paired to produce a stronger offspring. One of the heuristics in the pair must be a previously invoked heuristic. If a low performing heuristic was the one previously invoked, the quality of the weighted score outcome would be affected, since a weak heuristic is paired with a strong heuristic.

3. If a heuristic has already been deployed but becomes incapacitated due to different factors, including severe environment changes, there is no mechanism to move a search to another heuristic without breaking stride.

Hyper-heuristics have increasingly been developed with inherent dynamic capacity to address dynamic and complex problems [7]. The problem with today's search spaces is the shifting optima which is difficult to track by, among others, the methods that operate at the abstraction layer of hyper-heuristics. Therefore, a memetic algorithm (MA) is used to improve the search and reduce the risk of premature convergence. Furthermore, the search ability of MA is assessed and a new search mechanism is proposed to enhance its performance.

In order to address the presented shortcomings, three contributions are presented in this thesis. The first is an efficient choice function (ECF), which uses a new weighting score model in the selection of heuristics. In the proposed efficient choice function, the synergy between paired heuristics is aimed towards improving performance by ensuring that strong ones are paired. An obtained heuristic is then used to generate high quality solutions for various problems. Secondly, a new non-stationary covariance function is proposed as a predictive method for the Gaussian process regression to track shifting optima in a dynamic search space. Finally a spy-search mechanism is introduced to enhance MA's search efficiency in a dynamic search environment.

1.1 Motivation

Hyper-heuristics are considered as better methods for solving optimization problems than other optimization methods due to many factors [8]. One is the HHs' ability to sample heuristics rather than solutions. This factor makes HHs independent from problems so as to be applied to a wide array of them. However, the generic aspect of HHs requires a high degree of accuracy and efficiency so as to perform well on various problems. The fulcrum of hyper-heuristics is the heuristic selection process, where mechanisms such as a choice function are used [9] to choose a heuristic to map onto a problem. Therefore, there is an opportunity to improve the efficiency of HHs by focusing on a key feature that determines its overall performance.

Although there have been attempts to improve the structure of hyper-heuristics [2, 3, 10, 11], mostly focusing on improving parameters in perturbative hyper-heuristics, there are still further improvements that are needed to address the limitations in selective HHs, including the search mechanisms.

1.2 Problem Statement

Using generic methods to solve optimization problems is a challenge to modern search technology due to the adaptability and accuracy required to effectively solve today's dynamic and complex problems [12]. Moreover, today's problems keep evolving to even more complex and dynamic problems. Although there has been progress made in the field of HHs, most of the existing work has focused on static hyper-heuristics [13, 14], hence emphasis needs to shift to adaptive HHs that can adequately solve dynamic problems. In the heuristics' search space, a desired heuristic must be intelligently selected based on the information obtained from the problem domain such as, the problem representation, problem instances, evaluation function, low level heuristics, among other [15]. The re-usability of the selected heuristic should be an important part of HHs, since heuristics are not limited to a particular problem. Therefore, a choice function should have a storage component for heuristics and a mechanism on how they can be re-used.

The following two search methods are used in hyper-heuristics.

1. Single point search
2. Multiple point search

Single-point search methods generate a single outcome from a search, while multiple-point search methods generate multiple solutions from a search and evolutionary methods are applied to optimize the solutions [14, 16]. These methods mostly apply genetic-based search techniques [17] to search for optima and evolve heuristics through the evolutionary process. Given the increasing complexity of problem environments and the need for generic methods to be efficient on all problems, more emphasis should be placed on using memetic-based search methods because they combine both global and local search techniques. This will create a synergy between different search techniques in order to prevent premature convergence through intensification and diversification of a search. A synergy is created by merging different techniques to leverage their strengths. When the search process is enhanced through leveraging the strengths of different techniques, it becomes possible to track moving optima in dynamic search spaces.

1.3 Thesis Objectives

The main objective of this research is to improve the search and selection performance of HHs in optimization. The specific research objectives are:

1. To design a new search mechanism for MA search in dynamic HHs.
2. To introduce a new selection method called an efficient choice function for HHs.
3. To develop a new non-stationary covariance function for dynamic search in HHs.

1.4 Contributions of the Thesis

This research pushes the boundary of knowledge in HHs, specifically in areas of search and intelligent selection process through the following additions.

1. An improvement of a choice function selection method to enhance the performance of HHs.
2. A novel search method is introduced for MA called a spy search method (SSM) which improves convergence of the algorithm and search performance of HHs.
3. The design of non-stationary covariance function as a predictive measure for the Gaussian process regression to track shifting optima, thereby improving search performance of HHs in dynamic problem environments.
4. A review of the existing related work in MA and HHs.

1.5 Scope of the Study

In this research, focus is placed on the search and selection methods for HHs for both static and dynamic environments. HHs can use either single point search methods or multiple point search methods [16]. One of the examples of single point methods is a heuristic selection method known as a CF. In this study, more emphasis is placed on a choice function's heuristic selection mechanism, especially its weighted score model. The heuristic search conducted at the abstract layer of HHs is based on a MA and focus is therefore placed on improving its search capacity, using the proposed enhanced selection method. This research is also limited to dynamic HHs where the optima is not stationary. As a result, a method is developed to track the non-stationary optima in

dynamic problem environments. There are several different problems that are used to test the proposed approaches in this work. In addition, a hyper-heuristics flexible framework, which is used for testing new hyper-heuristics [18], is used in this study. Furthermore, statistical tools are used to perform analysis and the statistical significance of the results.

1.6 Thesis Overview

The organization of the rest of this thesis is as follows. Chapter 2 presents a review of the related work in HHs and MA. In Chapter 3, a CF for HHs is presented, including the proposed efficient choice function. The spy search method for memetic algorithms is covered in Chapter 4. The non-stationary covariance function for dynamic hyper-heuristics is presented in Chapter 5. Results and discussion are presented in Chapter 6. The conclusion and future work are presented in Chapter 7.

Chapter 2

Literature Review

2.1 Introduction

The concept of hyper-heuristics (HHs) dates back to the 1960s to the field of Artificial Intelligence and other related areas. The initial work hybridized different individual rules to increase performance in production scheduling [19]. The hybrid method showed better performance than the individual rules. The applied methodology was probabilistic learning, which made inferences from the available input information.

The main aim of HHs, as presented in literature [2, 20], is to choose heuristics from a set and map them onto problems. HHs can select a heuristic or generate new heuristics from the components of the available ones.

As demonstrated in Figure 2.1, there are different parts that form the main HHs structure and these are: problem domain, domain barrier and HHs domain. The problem domain encapsulates all the necessary information that pertains to the problem. The domain barrier is the middle part of the framework which sifts information crossing to the HHs layer so as to allow information that is not specific to the problem. The next part is the HHs domain where heuristics are generated. These are then mapped onto problems to find solutions.

2.1.1 Selection Hyper-heuristics

Hyper-heuristics can be categorized in terms of how they function. Selection HHs choose heuristics from a set. The methods used can be either constructive or perturbative, as shown in Figure 2.2. Constructive methods build the solution incrementally from an empty solution, while perturbative methods improve existing solutions [21].

As shown in Figure 2.2, heuristic selection HHs perform three types of learning. The first type is *online learning*, whereby information about the search space and problem

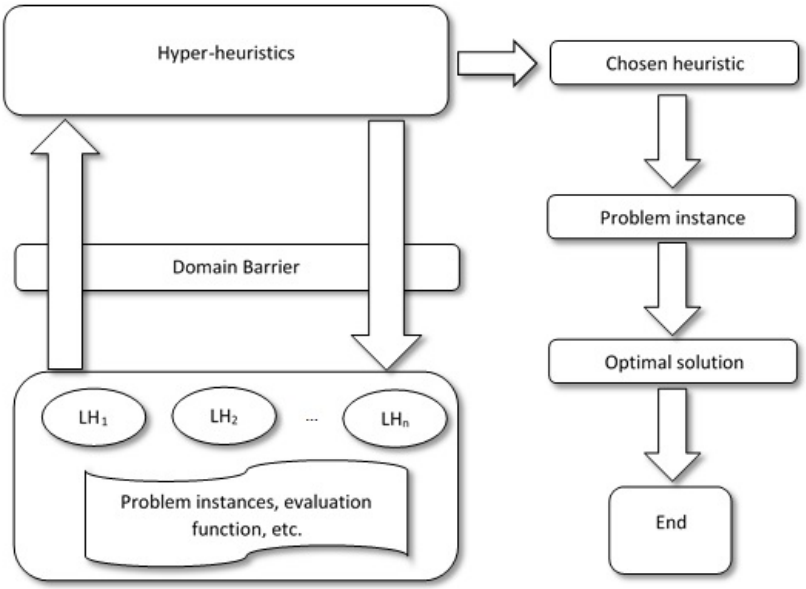

Figure 2.1: Hyper-heuristics simple structure

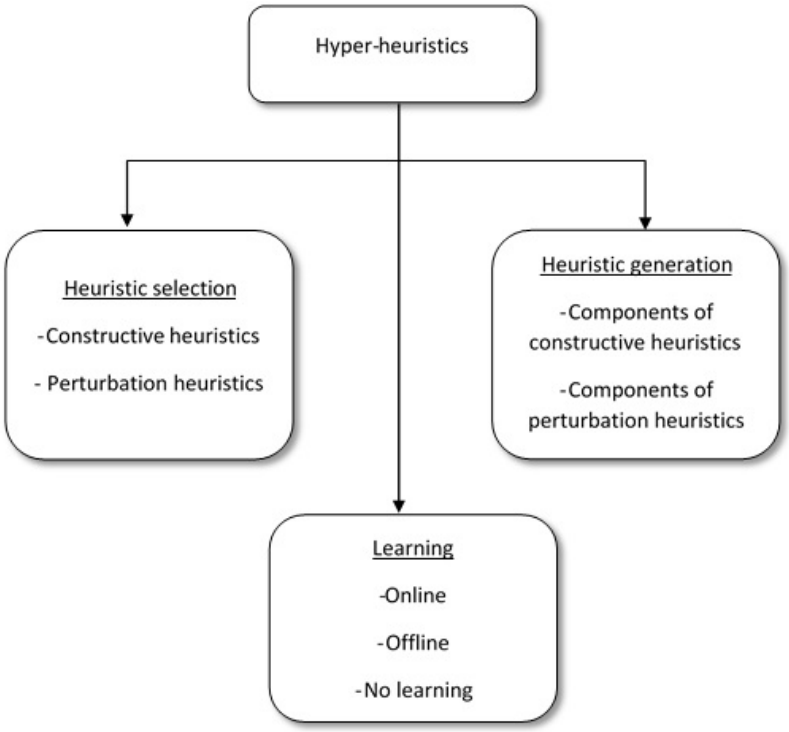

Figure 2.2: Hyper-heuristics classification

is gathered during the iteration while the problem is being solved. This information is used to find suitable solutions to problems. Another type is learning by offline, where knowledge is structured in the form of rules from available instances. The rules act as standards that must be observed by the search process. They help in obtaining specific information. In no learning, no feedback about the search space and problem environment is gathered from the search process.

2.1.2 Generation Hyper-heuristics

Generation HHs create heuristics from existing components [14]. The generation process follows the evolutionary structure of genetic programming which mimics a biological evolution, as shown in Algorithm 1. Genetic operators such as mutation and crossover are used to evolve new heuristics.

Algorithm 1: Genetic evolution

```
h←Initialize heuristics();
while stop conditions are not met do
    FitnessEvaluation();
    Choose parents based on fitness;
    Apply operators to produce offspring of heuristics;
    Evaluate offspring;
    Consider better outcomes;
    return best individual
end
```

In this category, generated heuristics can be either reusable or disposable. Reusable heuristics do not have a shelf life in terms of usage. They can be applied to as many problems as possible, whereas disposable heuristics can only operate on a single problem and cannot be reused for a different problem. The benefits of automating the heuristic generation process include time minimization, cost and error reduction, which in turn, cause an increase in efficiency of a heuristic [14].

2.1.3 Heuristic Selection and Acceptance Strategies

Heuristic selection methods and acceptance strategies are presented in [9,22,23]. They include, simple random, which randomly chooses a heuristic at every decision point on heuristic selection during the iteration. Random descent also randomly chooses a heuris-

tic, but focuses on the improvement of a solution. A heuristic is applied if an improvement in the solution is observed. Random permutation produces a permutation of heuristics in a random fashion and applies a heuristic at a time. The repeated application of a heuristic is done by the random permutation descent and the application of a heuristic stops when there is no further improvement. A greedy method chooses a heuristic that has yielded the best improvement in fitness. All heuristics are applied in order to find a heuristic that has generated a better outcome. Finally, a choice function uses a scoring technique to determine the best heuristic, as shown in Algorithm 2.

Algorithm 2: A choice function method

```
h←Initialize heuristics();
CF(h)← heuristic selected by a choice function while stop conditions are not met
do
    Rank h based on the score;
    Get CF(h);
    Apply CF(h);
    repeat
        Update the score and rank of h
        Choose CF(h)
        Apply CF(h)
    until;
    stop condition is met
end
```

In Algorithm 2, CF and h denote Choice Function and heuristic respectively. In most of the experiments presented in the literature, the choice function performs better than the other methods due to its scoring method [24]. A study by Bilgin et al. [25] shows that even after applying eight heuristic selection methods, a choice function always dominates in terms of performance. Many other examples on selection methods have also been reported in [23]. As a result, a choice function has become a common method for heuristics selection in HHs. In this research, emphasis is placed on the limitations of its scoring method and proposals to address them. Given the increasing complexity of today's problems and unstable search environments, optimization in HHs requires a more efficient heuristic selection method to intelligently choose the best heuristics and map them onto problems. MA is used to evolve heuristics and perform a neighborhood search for better outcomes in a sample of heuristics.

2.1.4 Memetic Algorithm

Memetic algorithm (MA) is a generic term for meta-heuristics that combine an evolutionary algorithm with other local search techniques so as to diversify and intensify the search. The synergy created from these approaches (global and local search techniques) reduce the possibility of premature convergence [26, 27]. MA uses a cultural evolution process to evolve solutions. The basic units of cultural evolution are called memes, which move from one individual to another through imitation. In Algorithm 3, as individuals interact in MA, memes are transferred from individuals to produce offspring.

Algorithm 3: The basic structure of a memetic algorithm

```
Initialize population;
while stopping criterion not satisfied do
    Evaluate every solution fitness;
    Select a pair of parents;
    Recombine individual parents to produce offspring;
    Mutate produced offspring;
    Apply the LocalSearch to improve solution;
    if offspring's fitness is better than parent then
        Replace parent;
    else
        Keep the old solution;
    end
end
```

The exploration and exploitation of the search are key features of a MA. However, there is no prior search that is conducted to establish areas which are more productive for a search. In addition, there is no mechanism to focus more resources in areas that would yield better outcomes. The imbalance in the search space in terms of fit individuals can be due to various reasons such as different fitness of individuals in different areas of the sample, environmental change, and so on [28]. Given that HHs deal with methods for problems, it is important that methods are evolved by considering all their behaviors and features through cultural evolution. Today's problems have become dynamic in nature due to increasing uncertainties, as presented by Jin and Branke [29]. As a result, the evolutionary techniques used should be able to adapt to the problem changes and moving peaks in dynamic search spaces.

2.1.5 Dynamic Hyper-heuristics

Dynamic HHs operate in dynamic search environments which are always changing. This implies that instances, objective functions, and constraints may be changed as the characteristics of an environment change [7, 30]. Changes can be determined in terms of their severity, frequency, predictability and cycle accuracy. The main challenge resulting from these changes is tracking the non-stationary optima during the search. An efficient HHs should be able to overcome these changes. A mechanism is therefore needed to make HHs able to react to the environment changes quickly, adaptively and predictably.

2.2 Literature Review

2.2.1 Hyper-heuristics

2.2.1.1 Selection Constructive Hyper-heuristics

Sabar et al. [31] proposed a constructive hyper-heuristic to develop a solution for an examination timetabling problem. One of the key features of this study was combining four low-level graph coloring heuristics to generate an ordered time schedule that meets the needs of students, for example, a difficult exam is placed first on a schedule. A probabilistic method was used to create time slots for each exam. A similar approach was presented by Ayob et al. [32], where heuristics were combined to solve a practical examination timetabling problem. The hybrid included largest degree first, saturation, biggest colored and biggest enroll heuristics. Although it performed well on a problem, it generated higher computational complexity when heuristics were combined.

Mlejnek and Kubalik [33] proposed a constructive hyper-heuristic and applied it to a capacitated routing problem. The mechanism was based on a local search algorithm named poems. Poems was used to iteratively improve the solution. However, the improvement was not applied to the constructed heuristic within the HHs. Aziz and Kendall [34] solved the same problem of capacitated routing by using an ant-based hyper-heuristic. In this study, the idea was graphically modeled with heuristics denoting nodes. Ants moved across the graph while leaving a trail with solutions. A probabilistic approach was used for ants to determine the next point of visit on the graph. This work was related to research presented by Ross and Marin-Blazquez [35], whereby operators were assigned equal probabilities and refinement was conducted continuously in the course of the iterations. The challenge with these approaches is that they do not cater for a changing environment where solutions are non-stationary.

Socha et al. [36] presented a constructive hyper-heuristic with a max-min ant system

utilized as a local search. This method was applied to solve a course timetabling problem which is defined with hard and soft constraints. Hard constraints are when a student cannot exceed one session attendance, a venue should have enough capacity for the number of students and each venue can take only one session at a time. Soft constraints are when a student can attend the last session, a student can attend multiple sessions in a row and the student has only one session per day. This problem was mapped onto the construction hyper-heuristic graph to create solutions. It used the graph construction elements proposed by Paechter et al. [37], which included time slot and corresponding objects such as venue, students, lecturers, and others. However, a local search improvement was missing in this approach, and yet it would be necessary, especially due to the infeasibility of some timetables. Performing a local search improvement would help to generate feasible timetable outcomes.

Furthermore, Pillay and Banzhaf [38] presented a study that solved the uncapacitated examination timetabling problem by constructing a timetable with a randomly selected allotted period. The approach addressed a problem of the process selection noise. One of its key features was that different timetables are designed and cost comparisons are made in order to ultimately consider the minimal outcome. The study applied the function that was proposed by Carter et al. [39], which helped to verify the effectiveness and quality level of the timetable. For example, an effective schedule made sure that the hard and soft constraints were strictly observed. This approach was related to the constructive hyper-heuristic that was proposed by Burke et al. [40], which also solved the problem of timetabling. This work used data mining to predict the sequence of heuristics. Another constructive HHs was proposed by Drake et al. [41], which was applied to the multidimensional knapsack problem. Every item has knapsack dimensions that fittingly take in resources and at every iteration a low level heuristic was applied. The approach was applied to unseen instances due to the generalization of heuristics, hence keeping heuristics blind to the problem information.

Thabtah and Cowling [42] solved a resource problem by applying a data mining method to select a heuristic to apply. The prediction method in this approach used the available data and rules to determine the behavior of heuristics. Furthermore, Chakhlevitch and Cowling [43] proposed a method for choosing the fittest subset of low level heuristics in a hyper-heuristic framework. The method considered a synthesis of different selection rules and was applied to the personnel scheduling problem. This approach was similar to the work presented by Corne and Ross [44], which combines greedy and random techniques to evolve the solution. However, these methods generated high computational complexity compared to work proposed by Abdullah et al. [45], which applied the roulette wheel to select the preferred low level heuristics. In this approach,

constructive hyper-heuristic found solutions for a problem of attribute reduction. Four constructive heuristics were considered and attributes were arbitrarily selected. One of key features of this approach was that it kept a history of the heuristics' data, which it used in its heuristics assessments. Rankhambe and Kavita [46] presented work that considered heuristics as a pair which used the matrix generated to define the pair and another matrix was defined to prevent premature convergence. Techniques proposed for heuristic selection included frequency based selection, count value updating and interval disturb. The frequency-based selection used the matrix in choosing the heuristic from a sample of low-level heuristics, while count value compares different objective functions for the past solution and new solution. The interval disturb was used mainly to ensure a search escape from falling into local optima. A hybrid of frequency-based selection, count value updating and interval disturb produced good results, although it generated high complexity and the performance of each heuristic was not adequately evaluated, especially in a dynamic context.

Jiang et al. [47] proposed a constructive hyper-heuristic where a new attribute was added to a semi-developed solution. The algorithm first produced a prototype of the solution and then used an iterative local search to perform necessary modifications. It is characterized by adding, modifying and deleting functions. These three functions were then applied to the prototype. This was similar to the approach proposed by Mlejnek and Kubalik [33], where the prototype was defined by the unit sequence which was constituted by constructive, two route improvement and single route improvement in order to produce a solution. The prototype was continuously modified until a better solution was selected. Furthermore, Garrido and Castro [48] proposed a constructive algorithm that created a partial set of solutions and then used the improvement heuristics to make refinements. The algorithm was applied to solve a capacitated vehicle routing problem. The problem with this approach is that if the solutions are weak at the start, further improvements will not produce desired results. It is important that methods be optimized before generating solutions.

2.2.1.2 Selection Perturbative Hyper-heuristics

In this area of HHs, the initial solutions are perturbed, starting with a full solution and then improvements are made to produce a better outcome. Refinements can be performed by perturbative techniques such as the swap, swift and others. However, changes may be rejected or accepted based on a move acceptance criterion. Raghavjee and Pillay [49] presented a selection perturbative hyper-heuristic for a school timetabling problem. In this work, parents were selected and then operators applied such as crossover and mutation. The tournament selection technique was used to select the parents and

violation mutation heuristics were used to ensure that constraints were observed. In case of any weak individuals, they were swapped with other strong individuals in the matrix using heuristics such as row, random and one-violation row swaps. These were used to swap items so that there were no clashes in the timetables. However, a more intelligent method is required to select individuals for swapping. Similar work was proposed by Maden et al. [50], which used a smaller set of low-level heuristics such as swap dimension that randomly selected solutions for swapping, hyper-mutation to overcome premature convergence of the search, dimensional mutation that randomly chose a single dimension and flipped the bits using genetic algorithm. Random descent, Simple random and Greedy methods were used in heuristics selection. This work used the genetic search instead of MA that would ensure diversification and intensification of the search.

Rae and Pillay [51] proposed a perturbative hyper-heuristic and applied it to a nurse rostering problem. This work addressed the challenges of high computational complexity observed in the work proposed by Burke et al. [52]. A multi-threading component was introduced which improved run-time process speed and efficiency. The work employed an evolutionary algorithm with a multiple point selection strategy. Different points were randomly selected and improved to arrive at a good outcome. Similar work was presented by Bai et al. [53], which used the stochastic ranking method merged with simulated annealing method to solve a nurse rostering problem. One of the weaknesses of constrained methods was the possible ruling out of the feasible regions, hence affecting the quality of the outcomes. However, this challenge was solved in the work proposed by Burke et al. [54], where the constrained problem was first converted to unconstrained problem and defined both feasibility and infeasibility functions to provide a balance of the model, in this way, the nurse details were evaluated based on their different shifts divided into day and night. The constraints in an evaluation function ensured equitable distribution of responsibility and minimization of errors. However, the methods did not show how they could operate in a dynamically changing problem environment.

Cowling et al. [2] applied their approach to a sales summit scheduling problem. The low-level heuristics applied a neighborhood search technique to a solution in order to produce a better one. Perturbation was done by the solution neighborhood search to improve it. A choice function was used to determine the next individual to consider based on the individuals' past performance record. One of the key features of this work was the synergy between intensification and diversification to enhance the quality of the solution outcome. Another study that focused on a perturbative hyper-heuristic was presented by Ozcan et al. [55], whereby two HHs were used in a successive fashion. Reinforcement learning was used for selecting the heuristic. Scores were given to low-level heuristics on account of performance and a heuristic with the highest score was chosen.

Ayob and Kendall [56] proposed a perturbative hyper-heuristic based on Monte Carlo. Every low-level heuristic improved its neighbor individual. The decision about the best solution was decided by the variants of Monte Carlo, created under its hyper-heuristic framework. These included linear, exponential and counter Monte Carlo methods. The counter method was specifically responsible for the diversification component, which ensured that the outcome was refined. Every variant of Monte Carlo used the probability value to accept or not accept the new solution produced by every low-level heuristic. This work was enhanced by Kendall and Mohamad [57], who determined methodically the low-level heuristic to be considered next in a succession of every point of decision. The approach was applied to a channel assignment problem. Different neighborhood methods were used to conduct the search, such as swap, delete and add methods. HHs was used to define the basis for accepting new solutions based on, record to record travel, accept all moves, only improving moves and Monte Carlo criteria. Bai and Kendall [58] presented a simulated annealing hyper-heuristic, which used parameters such as temperature to exploit the search. The method was applied to a shelf-space allocation problem, which is an extension of the multi-knapsack problem. Simulated annealing evolved individuals based on the problem. However, the parameters would not keep up with the changing search environment, thus requiring a more intelligent method for heuristic selection and search diversification.

2.2.1.3 Generation Constructive Hyper-heuristics

Generative constructive HHs construct new heuristics using the components of existent sets of heuristics. The characteristics from each heuristic are combined to build a new heuristic. Pillay [59] presented an approach that evolved heuristics to generate a new heuristic and applied it to a timetabling problem. In this work, genetic operators were used to produce new offspring at every iteration. Hill climbing was used to make solution improvements. Tournament selection method gave an element of competition of individuals, and a probability was set in order to provide a measure for choosing individuals that can become parents to mutate in order to produce offspring.

Burke et al. [60] simplified the heuristic generation process by making it automatic to reduce complexity. The method was applied to a bin packing problem. The fitness was determined by the bins available, the total of object pieces stacked within the bin and the bin's capacity. Genetic programming was used to evolve heuristics as reproduction and crossover operators were used to create new generations. Similarly, heuristics were evolved in the work presented by Kumar et al. [61], which was applied to a bi-objective knapsack problem. A key feature here was that each piece was evaluated based on two variables, weight and profit. If the outcome of the evaluation was less than one,

the iteration continued until a desired result was obtained and packed in the bin. The approach tried to trade off the weight and profit components by maximizing the profit while minimizing the weight. In order to evaluate the diversity, metrics were used such as hyper-volume, c-measure and convergence.

Furthermore, Pillay [62] proposed a hyper-heuristic for a one-dimension problem. The algorithm used two different methods, one for searching the space of heuristics and another for exploring combinations for item and bin. Tournament selection was used for finding parents for the mutation process, while hill climbing was used to improve the solutions by comparing the neighborhoods and then choosing better individuals. The set of heuristics used for selecting the bin included, first fit, best fit, next fit and worst fit. This meant that priority was given to the bin with enough space for an item and less residual space. If the bin was not enough, the next one was considered, and worst-fit considered the biggest residual space of the bin respectively. However, most of the reviewed methods under generation constructive HHs mutated individuals based on natural evolution, thus leaving out some aspects of individual behaviors and environmental changes which would be captured under a cultural evolution of individuals.

2.2.1.4 Generation Perturbative Hyper-heuristics

Generation perturbative HHs use genetic programming to generate heuristics and improve them through perturbation. Existing low-level heuristics are combined to form new ones, which are then perturbated in order to produce high quality heuristics. Edmund et al. [18] presented a Hyflex model framework that operated at a level of high abstraction. The framework was split into different layers which were tailored to ensure problem independence at the level of HHs. The problem domain consisted of, inter alia, evaluation function, low level heuristics and problem details. The domain barrier blocked problem-specific information from crossing to the abstraction level.

Vazquez-Rodriquez et al. [63] solved the problem of permutation flow shop, which processed a number of tasks in a consecutive order. The approach ensured that each task was processed one at a time, mapping it to one machine. A set of fifteen low-level heuristics was used and perturbation achieved by ruin and recreate and mutational methods. The steepest descent was applied to every job in the iteration to make local improvements. A new heuristic permutation could also be generated by using the crossover heuristic method, although it took long to arrive at the desired outcome. In order to address this challenge, Cowling et al. [8] proposed a tool for rapid prototyping in scheduling and optimization for HHs where a heuristic was chosen with a clear order to use for application. This is related to work by Chen et al. [64], which proposed an ant colony HHs. Ants moved as they applied a heuristic on every node in the path. The ants in this case

were called hyper-heuristic agents, as they moved towards the desired optimality. The low-level heuristics were improved through the perturbation by the ants on each vertex of the ant network. In another application of generation perturbative method, Han and Kendall [65] used guided operators and combined the encoding of genetic and MA to mutate low-level heuristics and enhanced the performance of GA. However, the methods did not cater for a changing problem environment with non-stationary optima.

2.2.1.5 Conclusion

This section reviewed some of the existing work in the literature to provide a context for the proposed ideas in this thesis. The purpose of HHs is two-fold. The first one is to increase abstraction for HHs so as to produce more generic heuristics. The second purpose is to adequately solve the problem at hand by generating good solutions. The review was conducted, based on a unique taxonomy of how HHs can be termed. The review also extended to different applications and domains that have been implemented in different methodologies of HHs. It is noted from the literature that there is growing interest in the field of HHs to solve hard problems. However, MAs have not been extensively applied in evolving heuristics. There are also less strategies for tracking moving optima in the search space.

2.2.2 Memetic Algorithm

2.2.2.1 Memetic Algorithm-based Hyper-heuristics

Ju et al. [66] applied a memetic algorithm (MA)-based HHs to both the permutation flow shop scheduling and personnel scheduling. The local search technique, which is a variable neighborhood search, was used to improve individuals in the population. The ruin and recreate method was applied to create a starting point for execution. Lei et al. [67] proposed a MA-based HHs that combined evolutionary strategy with a HHs for an examination timetabling problem. A list of graph coloring heuristics was used and operators such as mutation and crossover were used to create other heuristics. Once they were created, a local search method was used to optimize the solutions identified as feasible. In another work, a MA-based hyper-heuristic was applied to a multi-objective two-dimensional packing problem [68]. This approach was based on the first generation MA where a population global search method was reinforced with a learning mechanism. A parallel island-based model was used to reduce complexity and improve efficiency. Similarly, an island-based model was used alongside the hyper-heuristic in the work by Leon et al. [69]. The population was split into different sub-populations and a MA configuration was applied on each slice of the population. The approach was applied to

a bin packing problem and ensured that the search would not fall into local optima.

Smith [70] presented a co-evolving MA, whereby a local search technique monitored the patterns or repeated aspects in the sample space that was used to make the algorithm self-generating and self-evolving. This was in contrast to the grammar-based HHs framework proposed by Bader-El-Den et al. [71], where heuristics were evolved by the online learning method. Decomposition and grammar methods were used to decompose the heuristics into components that must be based on valid combinations. These components were then evolved into new heuristics. This approach was also used by Elyasaf [72] to evolve HHs. As noted, combining heuristics automatically created sub-problems which included, combining heuristics arithmetically, determining when to apply each heuristic or permutations, finding a fitting set of configurations that can enable the learning process to occur. Due to its complexity, Krasnogor [73] proposed a simpler way of evolution in MA, which simultaneously evolved individuals and local search operators that improved the solutions. The self-generating component was strengthened by the operators that were knowledge-based, mental simulation and imitation components. Consequently, new local search methods were generated. However, these methods did not include intelligent mechanisms of choosing a desired heuristic once improvements were done by the search techniques.

Merz and Freisleben [74] proposed a memetic algorithm that incorporated a new recombination operator to produce offspring of phenotypical composition. This work was applied to a quadratic assignment problem with 25 to 100 facilities in a similar way to the work of Merz and Freisleben [75], which used a new generic recombination operator for search exploitation. However, the methods were not adaptive to problem changes which could affect the search outcome in dynamic problem environments.

2.2.3 Conclusion

This section provided a review of some of the MA HHs presented in the literature. MA is nowadays preferred as a better evolutionary search method in hard optimization problems, especially for dynamic search environments. It is based on the synergy between global and local methods, which is leveraged to offset the weaknesses in each single method and, in turn, leverages, the strength of different algorithms to solve various optimization problems. However, more adaptive and self-evolving methods are needed for today's problems. It is also important to infuse the memetic evolution at the abstract layer so as to produce more abstract heuristics that can solve a wide range of problems.

Chapter 3

Proposed Selection Method for Hyper-heuristics

3.1 Introduction

Ever since Cowling et al. [9] introduced a choice function in the area of optimization, it has become a key component of hyper-heuristics (HHs). One of its key features is using a scoring technique to determine the ability of a heuristic. During the search, a score for every heuristic is maintained and a heuristic that generates the highest score is chosen. HHs use a move acceptance criterion to either reject or accept a selected heuristic [24]. A move acceptance criterion which expects the same outcome from the same set of defined solutions is known as deterministic and an example is all-moves, where a selected outcome is always accepted regardless of its ability to improve the solution or not. The only-improving criterion accepts the outcome if it is able to generate an improved solution. Improving and Equal accepts the outcome on two grounds: (i) if it can improve the solution, (ii) if it is the same as the current solution. A move acceptance criterion that expects a different outcome from the same defined solutions is known as non-deterministic. The outcome can be accepted if it improves the quality of the solution. However, a worse solution can be accepted using local search improvement techniques [23, 76, 77]. The move acceptance strategy is used in single-point search methods where a single outcome is generated. If many points are generated, for example, in multiple-point search methods, evolutionary methods are used to optimize the solutions.

In this chapter, focus is placed on a choice function (CF) which is a single-point search method for HHs. A heuristic is selected by a CF from a sample of heuristics at the abstract layer of hyper-heuristics. A CF is therefore a key method that hyper-heuristics use to decide which heuristic to map onto a problem [2]. The performance of HHs depends on the efficiency of a CF. For instance, a weak heuristic produced by

a choice function would not yield optimal outcomes when mapped onto the problem. Moreover, the dynamic nature of today's search spaces and the increasing complexity of problems demand a dynamic method that can adequately explore the search space for quality outcomes. In this chapter, the existing CF is investigated and a proposed efficient choice function (ECF) is presented.

3.2 Existing Selection Method

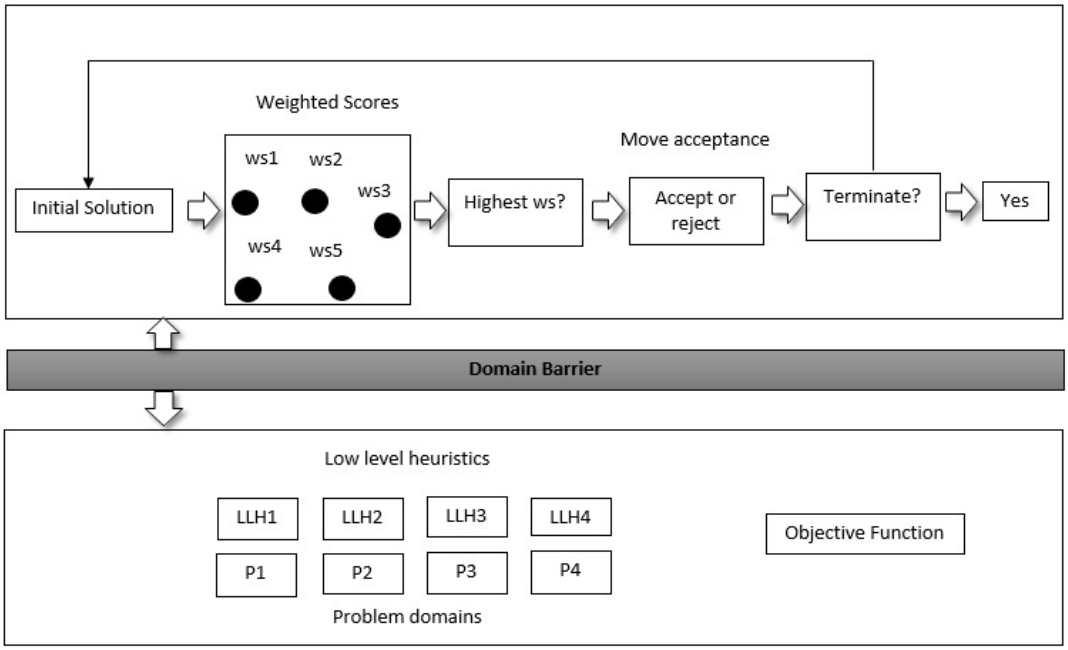

Figure 3.1: A choice function hyper-heuristic framework [18]

The problem domain consists of the objective function, low level heuristics, among others. Problem-independent information from the domain barrier such as random solutions, outcome of the objective function, among others, is allowed to the top layer of the hyper-heuristics framework in Figure 3.1. A CF based on a weighting score is then applied to find a desired heuristic. Once the best heuristic is obtained, a procedure for accepting the move is invoked. P and LLH denote problem domains and low level heuristics respectively, and while ws denotes weighted score. There are three components that

exist in the structure of a choice function, as defined in the following equations [9].

$$f_1(h_j) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \tag{3.1}$$

Where f_1 represents the previous performance value of a heuristic, $I_n(h_j)$ represents quality changes in a solution. T_n is the execution time for a heuristic. Parameters α and β in equations 3.1 and 3.2 respectively are used to intensify the exploration of the search through the adjustment of the parameter values.

$$f_2(h_k, h_j) = \sum_n \beta^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)} \tag{3.2}$$

Where (h_j and h_k) is the synergy between heuristics in a pair and the fitness of the heuristics is denoted by $I_n(h_k, h_j)$. The synergy is necessary to create an offspring that is representative of both parents' genotypes and phenotypes. The synergy also creates cooperation and interaction between heuristics in order to create high quality solutions to the problem at hand. f_2 represents a reward for heuristics that are consecutively applied, for example, h_k is applied after h_j. The fitness change and time for invoking a heuristic are represented by $I_n(h_k, h_j)$ and $T_n(h_k, h_j)$ respectively.

$$f_3(h_j) = \tau(h_j) \tag{3.3}$$

Where $\tau(h_j)$ in the third measure, f_3 represents time that elapsed since a heuristic h_j was last chosen by a CF. In Equation (3.4) δ is used to control the diversity of the search and ranking is then defined in Equation (3.4), based on the weighted sum of all the elements defined in Equations (3.1), (3.2) and (3.3).

$$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j) \tag{3.4}$$

There are a few improvements that have been made to the choice function. The first one is by Rattadilok [10], where the purpose of the modification was to allow the values of parameters to be updated automatically while keeping the hyper-heuristic problem independent. The modification by Drake et al. [3] focused on calibrating the intensity and diversity of the search such that in case of an improvement during the search, intensification was increased and diversification decreased. If there was no improvement, diversification would be increased and intensification decreased. Goncalves et al. [11] proposed the use of a scale factor and mean values instead of accumulated values for a choice function.

One of the drawbacks of the existing CF is inspired by the no-free-lunch theorem which was first presented by Wolpert [78] and improved by Schaffer [79] in machine learning. The no-free-lunch theorem says that the performance demonstrated by algorithms averaged over a set of problems is essentially the same for any algorithm. The assumption is that an algorithm may produce good results on a specific problem but perform dismally on a different one [80]. This implies that a heuristic that has shown a weak performance on a problem should be re-called because it may be a best heuristic for another suitable problem.

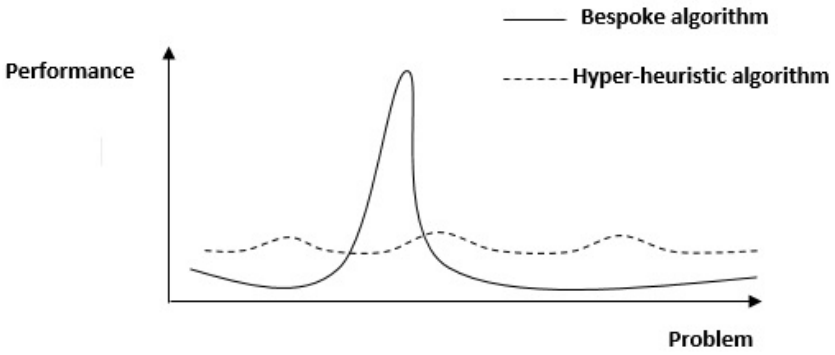

Figure 3.2: Algorithm performance based on no-free-lunch theorem

As shown in Figure 3.2, a tailored algorithm yields good results on a problem but returns poor results on another problem. The performance on y-axis increases as problem knowledge on x-axis increases in an algorithm. Algorithms that are not dependent on a problem show almost the same performance when averaged over a number of problems. It is therefore important that weak performing heuristics be stored and have a mechanism to recall them instead of generating new heuristics.

Another drawback of the existing choice function lies in the second measure f_2, which considers a synergy of heuristics that are ordered in a consecutive fashion. The problem with this technique is that a previously applied heuristic might be a poorly performing one, which then affects the quality of the output obtained from the synergy between a current heuristic and a heuristic that has recorded poor performance on a problem. This configuration causes premature convergence because of one strong performing heuristic previously invoked. The last area of concern in the existing method is that, ordinarily, optimization of heuristics stops when the iteration process starts. In the event of a heuristic's inability to continue the search due to changes in the problem environment

and problem complexity, there is no mechanism to move the search to another heuristic.

3.2.1 Proposed Efficient Choice Function

Three components of the proposed framework are presented in this work. In the first part of the solution, a storage for poorly performed heuristics is created, as depicted in Figure 3.3. During the iterations, a poorly performing heuristic is stored so as to be mapped onto a different problem. Heuristics are stored to be retrieved later for reuse. This is done using a mechanism as demonstrated in Figure 3.3. A heuristic is considered to be poor when it yields poor solutions for the problem at hand.

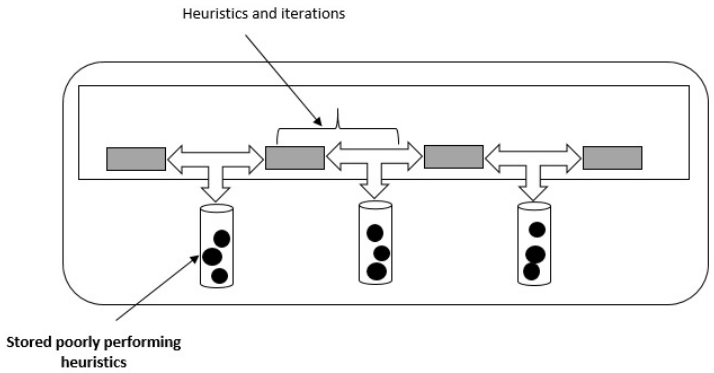

Figure 3.3: Storing low-performing heuristics in a proposed framework

Algorithm 4: Hyper-heuristics:heuristics storage

```
Create a ChoiceFunction object with a random seed;
Initialize solutions and supporting variables;
P_n ←Create a storage for low performing heuristics;
Initialize parameters α,β and δ;
Define a set of heuristics H_i;
while Termination is not reached do
    for int j=0;j<heuristics.length;j++ do
        if F[j] < highest_score then
            store_heuristic=P_n;
        end
    end
    for j = 0; j < number_of_heuristics; j++ do
        P[j] = phi * f1[j] + phi * f2[j][best_last_heuristic] + delta * f3[j];
    end
    Update the score of P(j), ∀h ∈ H;
    Select h with the largest F(i), ∀h ∈ H;
    Invoke the acceptance procedure;
end
```

In Algorithm 4, a set of heuristics is initialized and those that have yielded poor results are stored so as to be mapped onto another problem. The ECF scoring technique is continuously applied to obtain the weighted score for each heuristic. A heuristic with the largest value is chosen based on the acceptance procedure.

The second part of the proposed approach removes the existing consecutive pair synergy and introduces a synergy that is aimed at improving performance. The consecutive pair synergy is a pair method that follows a sequence of immediate next heuristic without any other focused criteria. Therefore, instead of pairing the current heuristic with the previous invoked heuristic consecutively, it is paired with the best heuristic in a pool of previously invoked heuristics. This is important to guide the pair wise process to produce a high quality outcome. The third component is to ensure that there is an alternative heuristic so that in case the performance is affected by, for example, changes in the problem environment, the alternative heuristic is invoked.

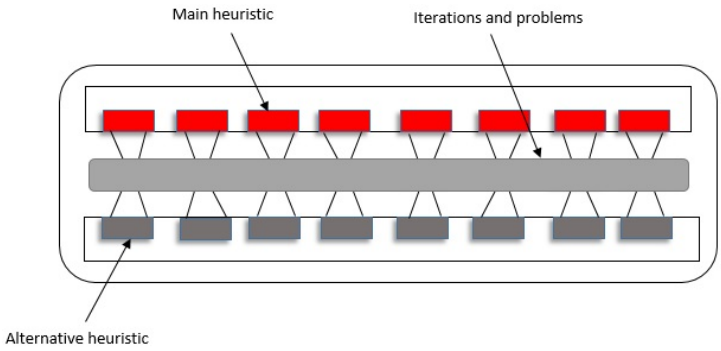

Figure 3.4: Proposed framework that maintains a pair-wise comparison at every point

Algorithm 5: Hyper-heuristics:heuristics pairing

Create a ChoiceFunction object with a random seed;
Initialize solutions and supporting variables;
Initialize parameters α,β and δ;
Define a set of heuristics H_i;
while *Stop criteria is not met* **do**
  for *int i=0;i>heuristics.length;i++* **do**
    $F[i]$ = phi * $f1[i]$ + phi * $f2[i]$[best_last_heuristic] + delta * $f3[i]$;
    if $F[i]$ = *highest score* **then**
      heuristic_to_apply = i;
      best_heuristic_score = $F[i]$;
    else
      $F[i]$ <highest_score
    end
    heuristic=h_j;
    Apply the selected i;
    Update the score of $F(i), \forall h \in H$;
    Select h with the largest $F(i), \forall h \in H$;
    Invoke the acceptance procedure;
  end
end

Algorithm 5 presents a continuous pair-wise comparison at every point in the course of the run. As demonstrated in Figure 3.4, in case of a heuristic's failure to adjust

to the changing problem environment, the alternative heuristic is ready to be applied to a problem at hand to generate solutions. The last part of the proposed framework is the collective performance of heuristics. Instead of considering the individual performance of each heuristic, low level heuristics are arbitrarily grouped into different clusters.

Algorithm 6: Hyper-heuristics:apply comparison at every point

```
Create a ChoiceFunction object with a random seed;
Initialize solutions and supporting variables;
Initialize parameters α,β and δ;
Define a set of heuristics H_i;
for int i=0;i>heuristics.length;i++ do
    F[i] = phi * f1[i] + phi * f2[i][best_last_heuristic] + delta * f3[i];
    if F[i] = highest_score then
        heuristics=h_i;
    else
        F[i] < highest_score
    end
    heuristics=h_j;
    Store in P_n;
    P[j] = phi * f1[j] + phi * f2[j][best_last_heuristic] + delta * f3[j];;
    if F[i] > P[j] then
        Select heuristic with F[i];
    else
        P[j] > F[i]
    end
    Select heuristic with P[j]
    repeat
        Update the scores of P(i) and P(j), ∀h ∈ H;
        Select h with the largest score in F(i) and P(j), ∀h ∈ H;
        Invoke the acceptance procedure;
    until Stop condition is met;
end
```

Algorithm 6 presents heuristic comparisons which are made at every point during the iterations. This is important to preserve competition and improve performance. Scores are updated for each heuristic and an acceptance procedure invoked in case of

any performance changes in either heuristic. h represents heuristic, P_n is the storage component, F is the fitness function, phi is the information collected for each element and delta is a diversity parameter.

The collective performance, as illustrated in Figure 3.5, is based on different clusters of heuristics, cluster A and cluster B. The best heuristic from cluster A is compared with the best heuristic from cluster B. Each pool of heuristics evolves into a new one by combining the strengths and attributes of the low level heuristics. The best heuristics obtained from each cluster go through another optimization process to find an optimal heuristic.

Let f_4 and f_5 represent measures of collective heuristics performance. f_4 and f_5 are new notations that have been introduced in the proposed method. Also, let f_1 represent the past heuristic performance, f_2 denote the synergy between the best heuristic and the current heuristic, f_3, indicate the time $[\tau(h_j)]$ since the last heuristic was invoked. q_t represents each point in the iteration. h_i is the best past performed heuristic. $P_t(h_j)$ represents storage for discarded heuristics. $I_n(h_j)$, $T_n(h_j)$ represent evaluation function changes and time spent to invoke the heuristic for each previous invocation n of heuristic, h_j. α, β and δ denote values that are between 0 and 1, that favor the most recent performance.

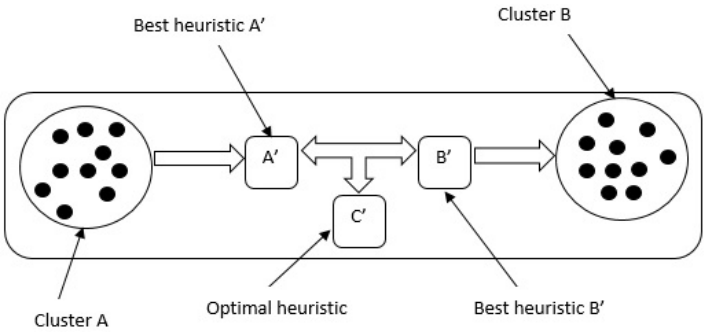

Figure 3.5: Proposed framework that is based on collective performance of heuristics

The ranking of heuristics in Equation (3.5) is obtained from the weighted score of different measures defined in these expressions, $f_1(h_j)\sum_n \alpha^{n-1}\frac{I_n(h_j)}{T_n(h_j)}$, $f_2(h_j)\sum_n \beta^{n-1}\frac{I_n(h_i,h_j)}{T_n(h_i,h_j)}$ and $P_t(h_j)$.

$$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_i, h_j) + \delta f_3 h_j \tag{3.5}$$

In addition, Equation (3.6) sums up all the weighted scores from different measures, including the weighted score from the collective performance measure. The heuristic with highest weighted score is preferred in the selection process of the heuristics in the sample. The inherent components that are introduced in this strategy contribute to the overall performance of the proposed efficient choice function. In order to apply a synergistic heuristic interaction at every point q_t, Equation (3.7) is invoked, which ultimately considers the maximum value from the synergy. In summary, the heuristic generated from individual process is compared with the heuristic obtained from the collective performance. $F_t(h_j)$ represents the first weighted score from the regular heuristics. $P_t(h_j)$ denotes a weighted score from the archived heuristics which were stored due to low performance. $F(h_k)$ defines a score from the different pools of heuristics in the sample for collective performance.

$$F(h_j) = F_t(h_j) + P_t(h_j) + F(h_k) \tag{3.6}$$

$$P_i F_j = q_t(h_i, h_j) \tag{3.7}$$

The overall weighted score, which aggregates different weighted scores, is defined as $F(h_j)$. This mechanism improves the quality of the outcomes and the parameters, as defined by Drake et al. [3], are left unchanged to weigh the measures defined in a choice function. Moreover, parameters control the two imperatives of the search, which are, intensity and diversity. The degree of diversity is increased if the search has not shown an improvement. Once there is an improvement, the intensity of the search is increased. The parameters are tuned to penalize low performance and offer a reward to improving heuristics.

Algorithm 7: Hyper-heuristics: heuristics collective performance

```
Create a ChoiceFunction object with a random seed;
Initialize solutions and supporting variables;
Initialize parameters α,β and δ;
Define a set of heuristics H_i;
for int i=0;i>5;i++ do
    groupHeuristics=A;
    F[i] = phi * f1[i] + phi * f2[i][best_last_heuristic] + delta * f3[i];
    if F[i] = highest_score then
        groupHeuristics=h_i;
    end
    for i greater than 5 and less than 10 do
        groupHeuristics=B F[k] = phi * f1[k] + phi * f2[k][best_last_heuristic] +
        delta * f3[k];
        if F[i] > F[k] then
            Apply A with F[i]
        else
            F[k] > F[i]
        end
        Apply B with F[k]
    end
    repeat
        Update the scores of F(i) and F(k);
        Select group with the largest score in F(i) and F(k);
        Invoke the acceptance procedure;
    until Stop condition is reached;
end
```

Algorithm 7 presents the clustering of heuristics. The number of heuristics in each group must be not more than ten and not less than five The combined strength of all heuristics is used to offset the weaknesses of individual ones. The weaknesses include low fitness, inability to adapt to environmental changes, among others. An efficient choice function scoring method is applied to find the best group with the highest weighted score.

The parameters can also be deployed adaptively, as presented by Cowling et al. [81]. The penalty and reward for the weighting depend on the degree of an improvement. This strategy helps to balance the diversity and intensification of the search space.

3.3 Conclusion

In this chapter, the existing choice function was discussed and its drawbacks were presented. The proposed ECF was also presented in this chapter. The effectiveness of selective HHs mainly depends on the efficiency of a CF. A CF intelligently makes a choice of a heuristic to be mapped onto the problem at hand. This study focused on single-point search mechanisms for HHs and four components were introduced to address the drawbacks of the ECF. The new method was tested on several problems and, as demonstrated in the experiments in Chapter 6, an ECF performs better than the existing methods. As part of the future work, all selection methods can be combined to offset their individual limitations. Also, predictive mechanisms for a CF can be introduced, especially in dynamic problem environments.

Chapter 4

Proposed Spy Search Method for Memetic Algorithm

4.1 Introduction

Memetic Algorithm (MA) is defined as the collaboration between cultural evolution and natural evolution to explore and exploit the search space [82]. In order to prevent a search from falling into local optima, a local search technique is used to improve the neighborhood of solutions in a sample space. A MA follows a cultural process to evolve individuals. This concept was proposed by Dawkins [26], who expressed that memes or units of knowledge can move from one brain to another. When memes reach a brain they evolve into new memes based on the nature of the current brain in terms of dispositions, attitudes, among others. However, today's problems have increasingly become dynamic. As a result, MAs are being applied in dynamic optimization where the behavior of elements keeps changing in the search space [83]. This affects the fitness of individual chromosomes thereby causing a drift in the location of the optima [83, 84]. In order to keep track of non-stationary optima and to optimally search the dynamic search environment, an intelligent search approach based on the search space information is required. There have been multiple strategies employed to enhance search quality such as, among others, intelligent search [85] and rule-based search [86, 87]. These mechanisms have been applied to various problems in optimization, but they have not performed well when applied to hard and evolving dynamic problems [88].

In this chapter, a spy search method (SSM) is proposed to improve the memetic search in dynamic problem environments. This approach is made up of different components such as hyper-mutation and random immigrants, fitness sharing and crowding and steepest mutation with greedy crossover hill climbing. The random immigrants and hypermutation are used to improve diversity, which is important for changing environ-

ments. A spy individual is dispatched to collect search space information. The method analyzes the information and uses it to perform the search. The approach is tested on four dynamic problems: oneMax, royalRoad, plateau and deceptive. The results indicate that an enhanced MA that uses a SSM performs well in dynamic search environments. The information collected before the search helps the method to direct more resources to areas that are more productive in terms of fitness and other features.

4.2 A Spy Search Method

The current search methods applied in optimization operate without first collecting information about the search space and its elements. The proposed SSM uses a spy individual to scope out the search space. A spy individual interacts with other individuals without revealing its identity to avoid being compromised and confused with other individuals in the sample. In addition, it does not keep individual profiles of solutions, but collects all information which is then analyzed, filtered and ranked so that it is deployed wisely in areas that are strategic and productive in terms of yielding high quality solutions. The identity of the individual among others is constituted by fitness value and positioning. The resources of the spy individual, such as memory, space, among others, are not shared due to concealment of its identity, hence keeping it superior in terms of fitness. The ranking is based on whether the information satisfies the parameters defined to guide information collection by a spy individual. The highest rank starts with fulfilling all the parameters, fulfilling some parameters and so on. The filtering is conducted to identify duplicated information and ensures that only relevant information is captured.

The benefits of pre-reviewing the search is strategic resource utilization to direct more resources to areas that are likely to produce promising outcomes from the known data. The algorithm is made up of different components. These are hybrid of hypermutation and random immigrants method, hill climbing local search, crowding and fitness and the steepest mutation with greedy crossover hill climbing. The motivation for using these methods is that these are common search techniques that have been extensively applied in different problem environments. They present different functions and merging them creates a robust platform that can be used to efficiently solve a wide range of complex problems.

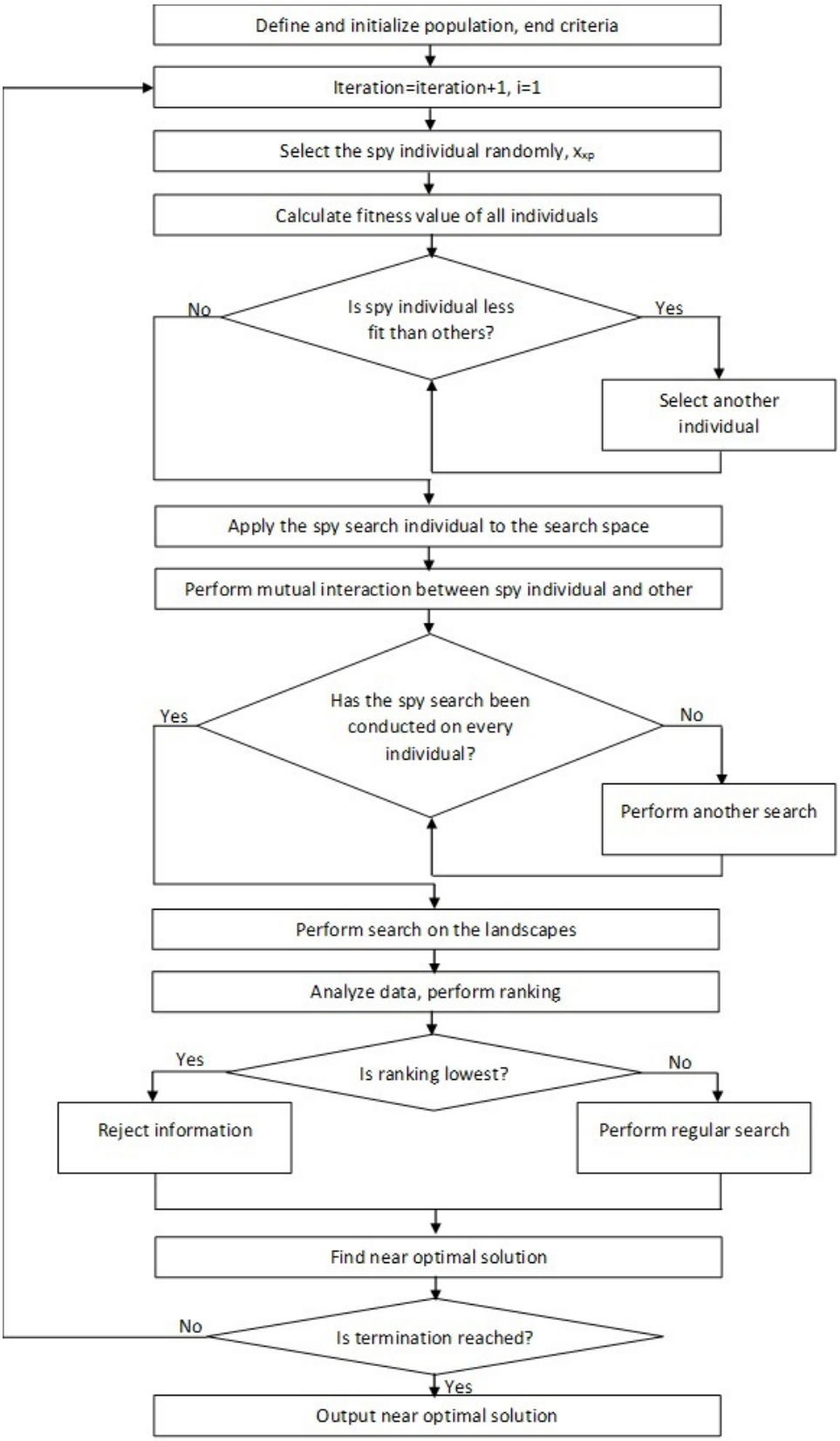

Figure 4.1: SSM flowchart

4.2.1 Hybrid of Hypermutation and Random Immigrants Methods

Hypermutation was proposed by Cobb [89] to balance the diversity level as the environment changes are happening. The method triggers an increase in mutation once there is a change in the search space. When the diversity increases, intensification decreases. This creates a balance between diversification and intensification, which are important in a dynamic environment.

In Figure 4.1, the population of individuals in the search sample is initialized. The fitness is determined for all individuals in the population so as to determine the baseline for selection of a spy individual. A spy individual is arbitrarily selected, but must have a strong fitness. If the selected spy individual is not fit, then another selection process is conducted until a strong one is obtained. Although hypermutation preserves diversity in the search space, Grefenstette [90] observed that when mutation remains high, convergence is affected. A random immigrants method replaces worst solutions with good solutions generated randomly, as shown in Algorithm 8. This method has been applied to solve many problems [90–92]. Immigrant individuals are those that move from one population to another in different generations.

Algorithm 8: Hybrid hypermutation and random immigrants procedure

```
Input:Individuals;
Output: Fit individuals;
Define parameters;
Initialize population, P;
Evaluate population, P;
while termination is not reached do
    P'(s):= reproductionSelection(P(s));
    Apply crossover;
    Apply mutation;
    Select random immigrants;
    Evaluate immigrants;
    if changes are detected then
        Perform hypermutation;
        Replace worst performing individuals;
    end
end
```

4.2.2 Hill Climbing

Hill climbing [93] operates iteratively starting with a random solution and then makes incremental changes. Better outcomes are then accepted to improve a solution as shown in Algorithm 9. The method is problem dependent [94] but it helps to prevent the search from falling into local optima [95, 96].

Algorithm 9: Neighborhood search

```
Input: parents;
Output: Offspring;
Define parameters;
Initialize population, P;
Evaluate population, P;
while Termination is not reached do
    NewIndividual← neighborhood(bestIndividual);
    if newIndividual is better than the currentIndividual then
        BestIndividual←currentIndividual;
    end
end
```

4.2.3 Fitness Sharing and Crowding

Niching mechanisms were first presented in genetic algorithms [97] to control the effect of selection operators in causing rapid changes to search space environments. A niche can be termed as an identity space within a search space. It is formed by individuals that have common properties. Fitness sharing adjusts the fitness of each individual to be equivalent to the sum of identical solutions in a search. Fitness sharing has been applied extensively in the literature [98, 99].

Crowding is where homogeneous individuals pooled from different generations are paired and then only one of the individuals is preserved during the replacement stage. As a result, this technique triggers an increase in individuals in areas that are underpopulated. DeJong [100] presented a parameter for crowding known, as crowding factor, which determines the candidate individuals for potential replacement. However, Casas [101] improved the method by focusing on diversity. A measure is introduced to manage niching such as giving information on the peaks existent in the population. Due to high complexity and unmitigated genetic shift, Mengshoel and Goldberg [102] proposed a new technique known as probabilistic crowding, which randomizes the selection process of the individuals so as to prevent a genetic shift towards a niche whose fitness is high.

4.2.4 The Steepest Mutation with Greedy Crossover Hill Climbing Technique

The greedy crossover hill climbing method is characterized by parents selected by the roulette wheel from a sample and a crossover is used in the production of an offspring, which later replaces the elite in case it has a superior fitness. The crossover can be single point or other types. In Algorithm 10, a steepest mutation changes the bits of chromosomes from the elite solution and an individual with better fitness is accepted [103].

Algorithm 10: Improvement with mutation

Input: Parents;
Output: Offsprings;
Define parameters, crossover & mutation probabilities Initialize population, P;
Evaluate population, P;
while *Termination is not reached* **do**
    Select parents $s = (i, j)$;
    Perform Crossover;
    Evaluate individuals;
    if *offspring, o is better than s* **then**
        Replace s with o;
    end
    Select individuals for mutation $s = (i, j)$;
    Perform Mutation;
    if *offspring, o is better than s* **then**
        Replace s with o;
    end
end

4.2.5 The Structure of a Spy Search Method

Considering a population of individuals, a spy individual is initialized as part of the sample, but with an aim of collecting information from the rest of the individuals, as shown in Algorithm 11. This includes fitness, positioning, nature of the environment, average time for environmental changes and their intervals, among others.
The distance between individuals helps to know how far apart the solutions are placed from each other. This information determines the distance changes resulting from the changes in the environment. The initial individual fitness also changes with environmental change. The positioning of the individual solution is determined by its locus and

Table 4.1: Notations used in this chapter

Distance	Fitness	Positioning	Past direction change	Change Severity
d_i	f_i	p_i	PC_i	ρ
Speed rate of Change	Diversity	Speed calibration	Spy individual	Population
τ	μ	δ	x_{sp}	P
Information gathered	Exclusive Operator	Offspring Population	Binary Template	index for environment
g	$\oplus$	P'	$\vec{T}$	k

environmental change patterns. The notations used in Equations (4.1), (4.2), (4.3) and (4.4) are defined in Table (4.1).

$$g_i = \sum_{i}^{n} (d_i + f_i + p_i + PC_i) \quad \forall i = 1, 2, ...n \tag{4.1}$$

The distance between individuals is calculated as a hamming distance, which is defined in [103].

$$d(x_{sp}, x_{i,...,n}) = \left[\frac{\sum_{i}^{n} |x_{sp} - x_{i,...,n}|}{n} \right] \quad \forall i = 1, 2, ...n \tag{4.2}$$

Also, consider that x_{sp} achieves its aim within the search by gathering data about the search elements $\forall i = 1, ..., n, (x_{sp}, i \geq g_i)$ where $\forall n = i + 1$ ensures that information is collected on each individual. Parameters control how the method reacts to changes in the problem environment. For example, when τ is increased, the technique is able to quickly find good individuals before the changes begin in the environment and the change severity is also balanced by parameter ρ, while δ is for speed. In addition, operators ensure that the search is sufficiently exploited [104].

Algorithm 11: Spy Search Memetic Algorithm

```
while Termination is not reached do
    if x_sp collects information then
        Store feedback information, Evaluate individual information;
        for Individuals in the search sample do
            if Fitness sharing and Crowding are used then
                Select similar individuals for reproduction;
                Pair individuals, Replace with a stronger individual;
                Adjust individual fitness to number of similarities in the crowd;
            end
            Calculate diversity,μ;
            if μ is normal then
                Apply LocalSearch;
            else
                μ is less;
                Spread out the population with Fitness sharing & Crowding ;
                Apply LocalSearch;
            end
            Select P′ for reproduction;
            Apply crossover, Mutation;
            Evaluate population, Calculate μ;
            if μ is normal then
                Apply LocalSearch;
            else
                μ is less;
                Spread out the population with Fitness sharing & Crowding;
                Apply LocalSearch;
            end
            if change occurs then
                Apply hypermutation;
                Triggered immigrants;
                Select the best individual;
            end
            if triggered immigrants is used then
                Apply immigrants to replace worst individuals;
                Apply LocalSearch;
                Select best individual;
            end
        end
    end
end
```

4.3 Dynamic Optimization Problems

Experiments for the approach introduced in this chapter are presented in Chapter 6 and they are based on the test environments generated by a dynamic optimization problems generator [105]. These functions have been extensively used to test new dynamic algorithms [103, 106–109]. The search environment is controlled by parameters τ and ρ which are defined as speed and severity respectively. When τ is at 0 the environment keeps intact, but changes when τ is at 1. In the binary string, the dynamic oneMax keeps the number of ones to maximum within it. The other functions contribute bits to the fitness of individuals, as presented by Forrest and Mitchell [110].

The results in this work were compared with the known results for different presented methods in the literature, which include memetic algorithm with adaptive hill climbing operator (AHMA) [111], deterministic dominance diploid genetic algorithm (DDDGA) [112], memetic algorithm greedy crossover hill climbing (CHMA) [111], genetic algorithm with random immigrants scheme (RIGA) [111], harmony search algorithm with random immigrant (HSA-I) [113], cooperative particle swarm optimizer modified harmony search (CPSO-MHS) [114], harmony search algorithm with memory mechanism (HSA-M) [113], multi swarm quantum particle swarm optimization (mQSO) [115] and harmony search algorithm with random based immigrant mechanism (HSA-MI) [113]. The justification for using the above methods for making comparisons is that most of them are dynamic and perform a memetic search in addition to combining different methods. They have also been widely used to solve dynamic optimization problems.

4.4 Conclusion

In this chapter, a new strategy for a dynamic MA was introduced. A SSM guided the search in a dynamic search environment by collecting information on individuals and the sample space environment. A greedy crossover was combined with the steepest mutation to make individual refinements. In addition, diversity was preserved by a hybrid of triggered random immigrants, hypermutation and crowding and fitness sharing, which improves convergence.

Experiments in Chapter 6 showed better performance of a SSM than the existing methods. The more severity in the changes, the stronger the performance shown by a SSM. As part of future work, multiple populations can be used and memory based approaches can also be explored. Change pattern recognition of the search environment by a spy search method would be another important area of research.

Chapter 5

Dynamic Selection Hyper-heuristics

5.1 Introduction

Hyper-heuristics (HHs) search within a sample of heuristics for a method that can be mapped onto a problem to produce good solutions. However, most HHs operate in a stationary search environment whose elements do not evolve with changes in a problem environment [19, 116]. As a result, most selection techniques in hyper-heuristics are applied to static optimization problems [117]. However, today's problem environments have become dynamic, hence they require adaptive methods to adequately solve complex problems. Dynamic HHs operate in a changing search space with moving optima. This implies that the objective functions and constraints continuously change as the problem environment changes [7]. Mechanisms are therefore required to make hyper-heuristics able to study the patterns of search environmental changes, make predictions and react accordingly. Hence, a new covariance function is created to operate with the Gaussian process regression so as to infer from the patterns of heuristics and enable the hyper-heuristic to make intelligent decisions.

5.1.1 Gaussian Process Regression

The Gaussian process is an encompassing term for probabilities distributed on functions. It has been widely studied and applied in many areas such as prediction. The Gaussian process regression focuses on the continuous prediction of the quantity set based on the continuous set of inputs. The inputs are typically generated from noisy calculations. The initial work of Gaussian process prediction was applied to time series as presented by Wiener [118] and since then its application has extended to other research areas. Gaussian process is also known as *kriging*, which originated in South Africa from an engineer in the mining industry [119]. The engineer applied the method in his spa-

tial mining work in determining dimensions and reducing complexities and inaccuracies of space measurements. The work later extended to statistics and other mathematical research environments that are especially based on multiple input variates.

5.1.2 Gaussian Process for Hyper-heuristics

Consider a random set, L of low level heuristics, $h_1, h_2, h_3, ..., h_n$ defined as inputs and the outputs corresponding with the inputs, $s_1, s_2, s_3, ..., s_n$. One of the presumptions made is noisy output observations from mapping of a function. Gaussian processes have been applied to obtain predictions from a given set of data and make inferences using a formula initially defined by Ebden [120]. Assume the objective is to predict a heuristics from a set L and the probability is denoted by $p(f)$.

$$p(f|L) = \frac{P(L|f)p(f)}{p(L)} \tag{5.1}$$

The regression therefore from the set $L = (h^i, s^i)_{i=1}^n$ can be defined as,

$$s^i = h(h^i) + \epsilon^i, \qquad i = 1, ..., n \tag{5.2}$$

The variables for noise are represented by ε^i with distinct distributions $N(0, \sigma^2)$ starting with a mean of zero. In addition, $\varepsilon[\epsilon(i\epsilon(j)] = \sigma^2\delta(ij)$ defines the noise variance and delta respectively for each low level heuristic as defined in Equation (5.2).

5.1.3 Non-stationary Covariance Function

Given that a mean for the Gaussian process is set to zero in this chapter, a covariance function is then used to relate different observations such as $k(x)$ and $k(x')$. T, N, L represent a test set of inputs, variables and low level heuristics respectively. The covariance between heuristics tracking of moving optima x' based on an array of low level heuristics are defined in Equations (5.3), (5.4) and (5.5) respectively.

$$cov(f(h_i), f(h_j)) \tag{5.3}$$

$$p(x, x') = N(0, L_{N+T}) \tag{5.4}$$

$$L_{N+T} = L_N \quad L_{NT}$$
$$L_{TN} \quad L_T \tag{5.5}$$

There have been numerous constructs of covariance functions applied to different problems [121]. Due to the continuous environmental changes, the method guides the search in a changing search space where elements are unstable, including optima, as shown in Figure 5.1. The navigation of the search can be defined as follows:

$$\mathbb{Q}_{ij} = (h_i - h_j)^T((\Sigma_i + \Sigma_j)/2)^{-1}(h_i - h_j)(h_i + h_j)\delta_{ij} \tag{5.6}$$

The kernel is denoted by $\sum_i$ and $\sum_i$ matrices at low level heuristics, h_i and h_j in the heuristic sample. The covariance function is also derived from the exponential format of the function and the matrices are left to evolve to a non-stationary function.

5.1.4 Gaussian Process Regression Prediction

The defined prediction is based on the covariance as well as the matrices and data points in the sample. In addition, the set of inputs for testing where the predictions are made is defined. The important components include the test points, outputs and training set. The correlations arise from the predictive pairs within the test set of variables. Correlations are useful for a variety of applications, since they are used to provide a measure of forecast uncertainty. The variance is defined in Equation (5.7).

$$p(\mu)_X = K_{XN}[K_N + \sigma^2]^{-1}i, j \tag{5.7}$$

$$\Sigma_X = K_X - K_{XN}[K_N + \sigma^2]^{-1}K_{i,j} \tag{5.8}$$

Both variance predictive expression and mean are constructed from the covariance defined in Equation (5.8) and at every point the variance and mean are applied to find the average variations on the heuristics sample.

5.1.5 Application to the Dynamic Traveling Salesman Problem

The various categories of dynamic optimization problems can be found in the work by Gharan and Saberi [123]. The dynamic stochasticity include moving optima and a

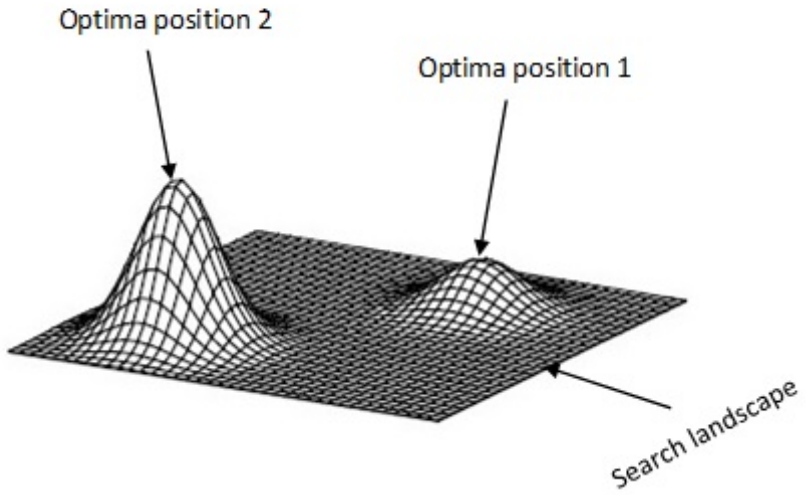

Figure 5.1: Optima in a non-stationary environment [122]

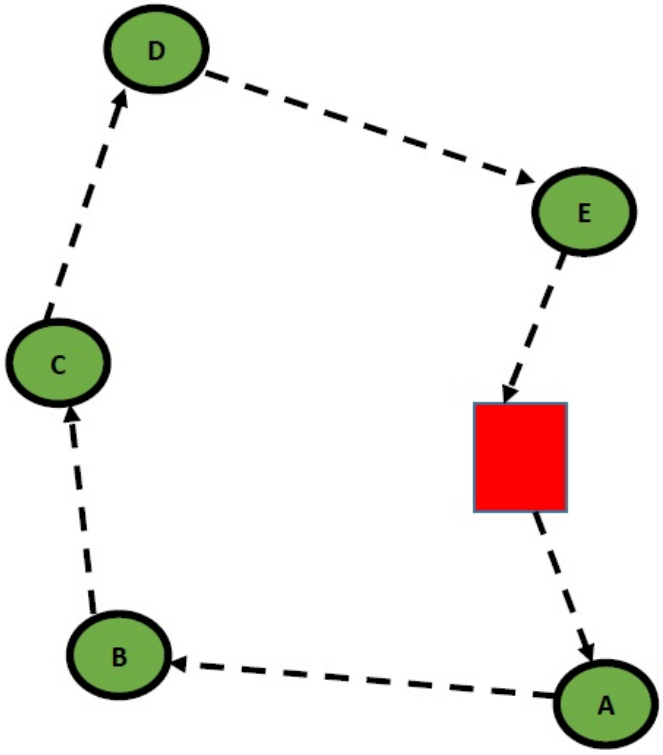

Figure 5.2: First request

changing problem environment. The optima keeps moving on a changing search landscape, which makes it hard to attain quick convergence, as shown in Figure 5.1.

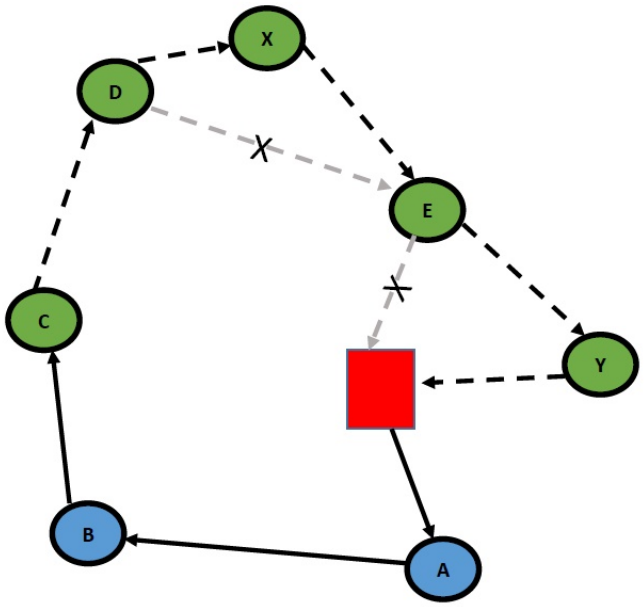

Figure 5.3: Changes occurred and new requests are received

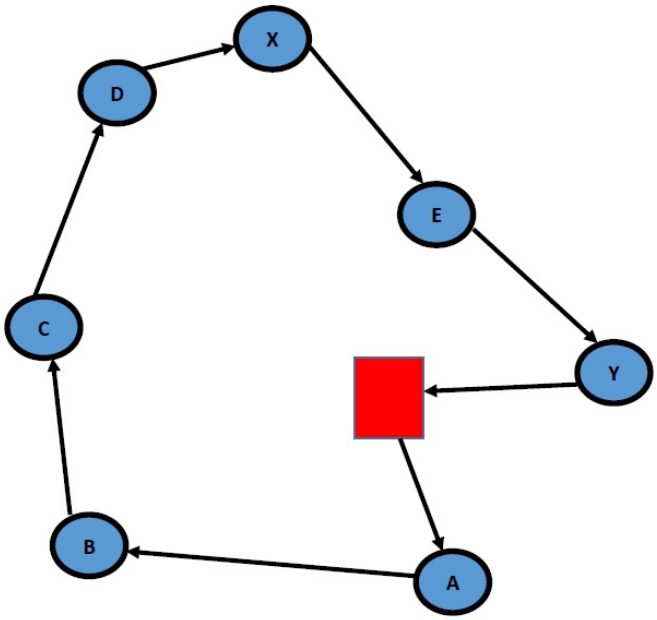

Figure 5.4: The path has been changed to factor the new requests

There has been a lot of work aimed at addressing dynamic problems and ensuring that the problem encoding fits the dynamic complexity of a problem [124]. The initial work in the dynamic traveling salesman problem (DTSP) was presented by Psaraftis [125] and Li et al. [126]. The typical function of the traveling salesman problem is to hop

from city to city once and then back to the starting point in the itinerary. The main feature of the dynamic traveling salesman problem is the ability to add and delete the city routes based on changing conditions. The goal is to reduce the distance traveled without diminishing the service of a traveling salesman, as shown in Figures 5.2, 5.4 and 5.4. Also, the objective is to find the path that has the shortest weighted distance. The problem has been modeled extensively in different optimization areas such as in work by Ray et al. [127]. Consider n, e, i, j and D as city sample, problem, start city, end city and distance respectively. The Euclidean distance between two cities is expressed as follows:

$$D_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{5.9}$$

5.1.6 Predictive Objective Function for the DTSP

Consider an array of costs for distance in the distance matrix of a sample city population, $(P_1, P_2, ..., P_{n(t)})$. The cost matrix is directly dependent on time, given that in the changing environment it is a big factor. Therefore, time remains a key function of the cost matrix. Assume that $D = d_{ij}(t)$ and $i, j = 1, 2, ..., n(t)$. The notations of time and distance are t and d respectively. Therefore, the minimization of the dynamic traveling salesman problem is achieved through the equation as follows:

$$Minimize \;\; d(T(t)) = \sum_{i=1}^{n(t)} d_{T_i, T_{i+1}}(t) \tag{5.10}$$

The first path of the dynamic traveling salesman problem is mapped out based on the traveling salesman visits constructed as $[A, B, C, D, E]$ in Figure 5.2.

Moreover, after the salesman has set off, new demands surface, which require a change of plan set down by a salesman. These changes are represented by (X, Y) and the new itinerary is set down as $[A, B, C, D, X, E, Y]$, as shown in Figure 5.3. The new requests are then incorporated, as shown in Figure 5.4. This adhoc plan reflects the continuous dynamic requirements of the dynamic traveling salesman problem. The changes must therefore be factored in the new route set for the trip.

5.2 Conclusion

This study applied the non-stationary covariance function together with Gaussian process regression for HHs. The proposed method was tested on the dynamic traveling salesman problem, whose environment changes from one state to another. One of the key features of the approach is the prediction of the tour and a near optimal solution

in a sample of cities. The dynamic approach helps to mitigate the impact of volatility by making estimations that are in tune with the search environment changes. Results in Chapter 6 show better outcomes generated by the proposed method, reducing cost and distance traveled for a traveling salesman even under changing search environments. As part of future work, more research can be directed to study the behavior of search environments and creating methods that can mimic, and react more effectively, to these changes.

Chapter 6

Results and Discussion

6.1 Introduction

The hyper-heuristics flexible framework (Hyflex) platform [18] was used to test some of the proposed methods in this thesis. Java and MATLAB environments were also used to perform the experiments. The Hyflex framework was developed to test new hyper-heuristics (HHs). It encapsulates sufficient problem information required for experiments such as problem instances, objective function, operators, solution representations [15]. The main class in a hyper-heuristics is extended to implement the proposed efficient choice function. This is possible due to the generic structure of Hyflex. The acceptance procedure is embedded in the framework with a choice function to accept moves based on the defined constraints. Hyflex also embeds the application components necessary for processing the objective function and for solution encoding, as shown in Algorithm 12. Two of the main classes of the framework are problem domain class and hyper-heuristics class, which divide the modular interface of the framework into different layers such as problem domain, domain barrier and hyper-heuristics layers. The framework also allows sharing computational load amongst different instances, depending on the time needed for each instance. Hyflex has a learning component, for example, if instances are in the same sample domain then a hyper-heuristic is able to learn from past performances of these instances. Hyflex also consists of various problem domains that can be used for experiments.

Algorithm 12: Hyper-heuristics Flexible Framework

```
h←Initialize heuristics();
s←Initialize solutions();
f←Define Objective function();
while Time not yet expired do
    if Information,f is not problem dependent then
        f ← Allow through the domain barrier to top layer
    else
        Reject f at the barrier;
    end
    Apply Heuristics Selection Method;
    if Best h then
        Select Heuristic;
        Apply Heuristic;
    end
    return Best Solution
end
```

6.2 Problem Domains

6.2.1 Traveling Salesman Problem

The goal of the traveling salesman problem (TSP) is to visit every city in the shortest time and distance possible, and return to the starting location. According to Lenstra and Kan [128], for N number of cities and the cost distance between cities $c_{ij}...(i, j \in N)$,

$$min_{\pi} \sum_{i \in N} c_{i\pi(i)} \tag{6.1}$$

The salesman reaches every k^{th} city at the $(\pi)^k(i)$ from the starting point i. The problem is then presented as follows:

$$min_{v}(\sum_{i=1}^{i=n-1} c_{v(i)v(i+1)} + c_{v(n)v(1)}) \tag{6.2}$$

The salesman moves from city to city at a cost of c_{ij}. The problem can be either symmetric, asymmetric and multiTSP. Symmetric cost means the distance is the same between different cities. Asymmetric costs are costs between cities that are at different distances,

$c_{ij} \neq c_{ji}$. The traveling salesman problem has been widely applied in various applications [129, 130]. The TSP instances used in Hyflex are from Reinelt [131].

6.2.2 Vehicle Routing Problem

The vehicle routing problem (VRP) was derived from a truck dispatching formulation [130]. In this work, the model is based on distribution of oil to various gas stations using a fleet of trucks. The problem was improved in 1964 [132] to include a number of customers that are in different geographical locations being served by different capacity trucks from a depot. The problem has now expanded to consider diverse vehicles within the fleet and different intervals of time. The vehicle routing problem instances used in this work are obtained in [133]. The heuristics used can be either constructive, improvement or order. The constructive heuristics for the vehicle routing problem include sweep, route first, route second cluster methods, as defined by Laporte et al. [134]. The improvement heuristics ensure refinement until there is local convergence. The improvement heuristics include 2, 3 opts, single and double routes. The order category of heuristics for the vehicle routing problem is based on changes in demand and distance between the depots in distinct geographical locations.

6.2.3 Personnel Scheduling Problem

The goal of the personnel scheduling problem (PSP) is to minimize time for personnel tasks and reduce the cost for labor, thus causing a reduction in the number of staff. The tasks or activities are organized into time periods. $A_i(t)$ is the staff required for a task i within t time. Every task i has a time limit. The staff, s, is given a working schedule, l, and the cost is represented by o. Let S be the set of n number of staff and D be the time duration. y_{sl} is the variable that is set to 1, subject to the staff being allocated the working schedule. o_{sl} represents the cost, subject to the staff being allocated the working schedule.

$$min \sum_{s \in S} \sum_{l \in D_s} y_{sl} o_{sl} \tag{6.3}$$

This is subject to the constraints as follows:

$$\sum_{l \in D_s} o_{sl} \leq 1, s \in S \tag{6.4}$$

$$\sum_{s \in S} \sum_{l \in D_s} l(i,t) o_{sl} \geq A_i(t), \forall (i,t) \tag{6.5}$$

$$o_{sl} \in 0, 1 \forall s \in S \; and \; l \in D_s \tag{6.6}$$

The constraints ensure that each staff gets one working schedule and is assigned an equal number of tasks. As defined by Dowsland [135], the objective is to get the best schedule with the lowest penalty costs, labor and staff. Instances used for this problem were obtained from Curtois [136,137]. Hyper-heuristics was applied to tackle the problem of personnel scheduling in the hospital context on nurse scheduling [138]. Instances were obtained from three wards in a UK hospital and comparisons are performed against tabu search and genetic algorithms. In order to generate feasible solutions the work was divided into balanced shifts of both day and night, and different parameters were defined, as presented by Cowling et al. [138].

6.2.4 Permutation Flow Shop Problem

The goal of a permutation flow-shop problem is to determine a sequence of job processing in the corresponding machine sequence. The machines process jobs in a successive fashion and only a single job is allowed entry into a machine. In this problem, the constraints ensure that the queue is followed, since every job waits for its turn and every machine keeps active or ready to take in the job. Permutations are generated based on the schedule for each job, especially time, when the job will begin and when the job is ending. Let $\theta = \theta(1), ..., \theta(n)$. $\theta(i), j$ represent job and machine respectively. Every job to be processed is allotted time and space on the machine, which is defined by e_{ij}. Moreover, the time when a job will complete is also set and denoted by F_i on j machine, as defined below.

$$F_{\theta(i),j} = start_{\theta(i)} + F_{\theta(i),j} \tag{6.7}$$

There are multiple orders of processing i jobs. The task is to find out the order that will take the least time to complete. The problem starts with a random generation of job permutations. After forming a schedule, the first job in the queue is assigned a schedule followed by the next job until all jobs have been allotted schedules. The instances used for this problem were obtained from Taillard [139]. In Hyflex, different perturbative heuristics are employed by perturbing solutions to generate better permutations of the solution. For example, Ruin and recreate reconstructs sequences of jobs after dismantling the initial order. This heuristic has been applied to a permutation flow shop problem and other related problems such as the greedy solution improvement [140].

6.2.5 Bin Packing Problem

The objective of the bin packing problem is to minimize the number of bins that can be used to load the available items, as shown in Figure 6.1.

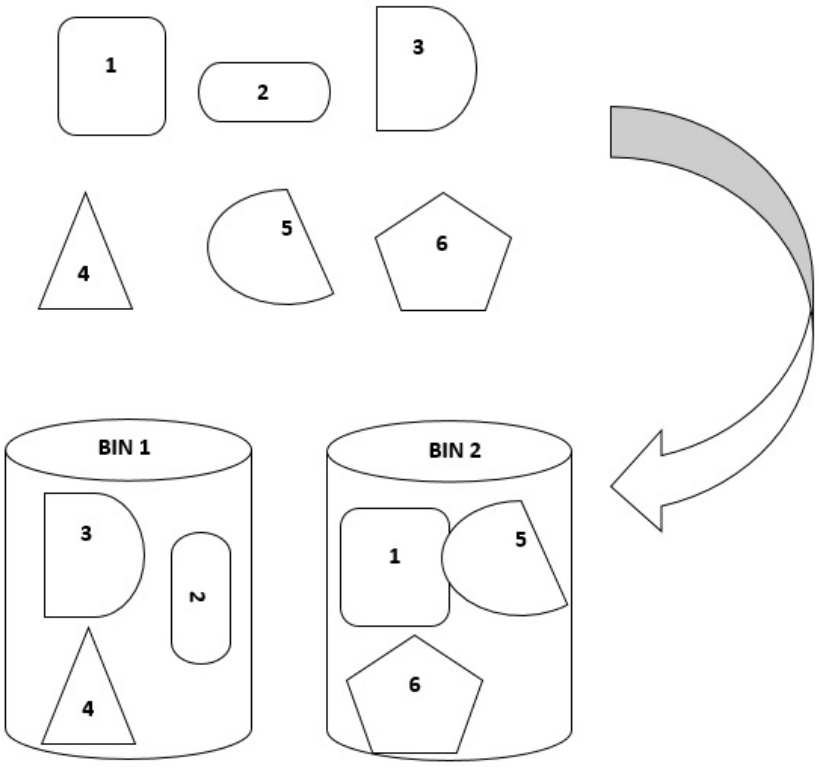

Figure 6.1: Bin packing problem

The items are different in weight and one of the conditions is that the weight of all items must not go beyond what the bin is supposed to maximally take. A bin packing problem can be defined as follows [141]:

$$Min \sum_{i=1}^{n} z_i \tag{6.8}$$

This is based on the constraints as follows:

$$\sum_{j=1}^{n} v_j p_{ij} \leq b z_i, \qquad i \in N = 1, ..., n \tag{6.9}$$

$$\sum_{i=1}^{n} p_{ij} = 1, \qquad j \in N \tag{6.10}$$

$$z_i \in 0, 1, \qquad i \in N \tag{6.11}$$

$$p_{ij} \in 0, 1, \qquad i \in N, j \in N \tag{6.12}$$

p_{ij} and z_i determines whether item j is in the bin and whether the bin is not empty, respectively. Bins that are not yet full and can still accommodate some items are considered [142]. A perturbative hyper-heuristic was proposed by Burke et al. [143] to solve a bin packing problem. The approach used genetic programming in evolving heuristics and perturbing them to produce an algorithm that can produce good solutions.

6.2.6 Patient Transportation Problem

The Patient Transportation Problem (PTP) is a popular problem in optimization, which also falls under the dial-a-ride classic set of problems. A dial-a-ride represents a transportation problem whose objective is picking up a number of clients from a starting point to a specific destination [144]. The data sets used were obtained in [145]. The problem consists of, among others, vehicles to be used and a set of constraints to control the process, as defined by Jorgensen et al. [146]. Time windows are used to control the requests and define the period that vehicles take to execute their assignments. The problem of patient transportation starts with a request from a patient, which consists of a pickup location, destination, the time period within which the patient can be picked up and delivered, and the transportation type to use based on the condition of the patient, such as ambulance, wheelchair or other. There is a limit attached to time so that a request is served in the required time. The patient may also be barred by their medical condition from sharing the transport facility with another patient to avoid possible contagion of the disease. The dynamics of the problem expand to other critical medical devices that could be needed in the transport facility as well, such as oxygen equipment and other medical gadgets for monitoring patients [146].

The requests from patients are denoted by r. The initial point for picking up the patient is represented by p, while the destination point is denoted by $r + p$. The patient can be picked up within the time range $[x_p, y_p]$, where x_p is the minimum time, while y_p is the maximum time, and the range for delivery x_{r+p}, y_{r+p} is defined based on the needs of the patient. s_i is the time needed for the medical service. The mode attached to the

Table 6.1: The problem notations for patient transportation used in this work

Notation	Interpretation
w_a^p	The time the patient has to spend waiting before using transport facility a
L=1,...,n	This denotes the locations where the patient can be picked up by the transport facility.
R=n+1,...,2n	The set of delivery points
N=K ∪ E	A set of points for both picking up patients and delivering them.
HRT	The highest duration placed on the route
C	Motor vehicle pool
V	This represents a pool of vehicles used in the solution.
HDT	Highest driving time in terms of how long the difference should be between picking up a patient and delivering the patient to the destination
t_{ij}	This indicates the time it takes from i to j locations.
l_i, l_j	The changes in the load of vehicles at different locations
$B = N \cup$ g	Potential stopping points for the vehicle
$t_{p,q}$	The distance or time taken to travel from one location to the other
s_p	The needed time for servicing the request
Δt_r	The allowed deviation from the defined time for service of request r
EP_r	This is the earliest time the patient can be picked up.
z_{ij}^c	The variable with 1 as the value if the facility c handles patient on point i then followed by another patient at point j.
T_i^c	The time for the service facility to begin
T_j^c	The time when the facility ends its service
O_i^c	The load for each facility after visiting i location
G_i^c	This is the waiting time for the facility before servicing point i.
LP_r	This represents the latest time the patient can be picked up after making a request.
ST	Time when considering direct distance between points of pickup and destination putting into consideration dwell moments
$C_{v,m}^k$	The capacity of the transport facility v and m as mode of transport and k for the alternative

request, r_p^m, can be w for wheel chair, s for seating transportation, e for stretcher and d for bed. Each facility is generally denoted by b. Also, in terms of equipment, $w_p^o = 1$ if

the medical equipment o is needed and if it is 0 then it is not needed. If there is no need of vehicle sharing for medical reasons, $y_p = 1$ is invoked and $y_p = 0$ if patient isolation is not required. $_p$ represents the disinfection period set for the exercise. There are a transport facilities available for meeting the transport needs of patients and these have a specific capacity denoted by L. Every transport facility begins and stops at different schedules represented by T_b^z and T_r^z respectively from p location. The rest of the notations are summarized in Table 6.1.

Dwell includes time spent by patients getting into and out of the vehicle at both initial point and destination. It also includes the extra time spent at the points before boarding and during disembarking. The desired period for both picking up and delivering procedures are defined. Moreover, given that the proposed efficient choice function is used, as presented in Algorithm 13, a set of weights are used to generate weighting scores represented as l_n for the objective function, for example, l_1 denotes the weight on the duration of vehicles as they move between different points. l_2 is the weight on lateness of a facility. l_3 weights time taken by patients to wait for the transport facility, while l_4 weights the path duration defined by the period between the start of the service to the end of the service. The penalties are applied to schedule violations and weighted by l_5. The objective function used is expressed as follows:

$$Minimize \;\; l_1 \sum_{c \in Y} \sum_{i,j \in B} t_{i,j} z_{i,j}^c \tag{6.13}$$

$$l_2 \sum_{c \in Y} \sum_{i \in L} T_{n+1}^c s_i (T_i^c - t_{i,n+1}) \tag{6.14}$$

$$l_3 \sum_{c \in Y} \sum_{i \in N} G_i^c (O_i^c - l_i) \tag{6.15}$$

$$l_4 \sum_{c \in Y} \sum_{i \in L} max(0, (T_{n+i}^c - T_i^m) - HDT) \tag{6.16}$$

$$l_5 \sum_{c \in Y} max(0, (T_i^c - T_j^c) - HDT) \tag{6.17}$$

This is subject to

$$\sum_{c \in V} \sum_{j \in L} x_{i,r}^c = a \tag{6.18}$$

$$\sum_{c \in Y} \sum_{i \in R} x_{j,r}^{c} = a \tag{6.19}$$

$$\sum_{j \in B} x_{i,j}^{c} - \sum_{j \in B} x_{j,i}^{c} = a \forall c \in V, i \in N \tag{6.20}$$

$$\sum_{c \in V} \sum_{j \in N} x_{i,j}^{c} = 1 \quad \forall i \in L \tag{6.21}$$

$$\sum_{j \in N} x_{i,j}^{c} - \sum_{j \in N} x_{j,n+1}^{c} = 0 \quad \forall c \in V, i \in L \tag{6.22}$$

$$O_{r}^{c} = 0 \quad \forall c \in V \tag{6.23}$$

$$x_{i,j}^{c} \in 0, 1 \quad \forall c \in C, \ i, j \in B \tag{6.24}$$

$$T_{i}^{c} \geq 0 \quad \forall c \in C, \ i \in C \tag{6.25}$$

In Equations (6.18) and (6.19), a transport facility at the hospital starts and finishes at different locations of the defined schedule. In order to ensure that there is an equal number of facilities providing the services, Equation (6.20) is invoked, while Equation (6.21) forces the facility to serve a location at a time. Moreover, the pickup location of the patient should be visited prior to the destination location point and this is enforced by the Equation (6.22). In order to monitor the transport facilities to ensure that they are empty before starting and after their activities, Equation (6.23) is used. In addition, Equations (6.24) and (6.25) respectively ensure that the same transport facility can pickup patients from the source and deliver them to their delivery point, and ensure that the space capacity for the facility is observed.

Algorithm 13: The pseudocode for the application of the proposed choice function to the patient transportation problem

```
Create a ChoiceFunction object with a random seed;
Initialize solutions and supporting variables;
Create Storage, P[j];
α, β, δ ← Parameters;
F, f1, f2, f3 ← Weights;
H_i ← A set of heuristics;
for int i=0;i>heuristics.length;i++ do
    F[i] = phi * f1[i] + phi * f2[i][best_last_heuristic] + delta * f3[i] if F[i] =
    highest score then
        heuristic=h_j;
        heuristic_to_apply = i;
        best_heuristic_score = F[i];
    else
        F[i] <highest_score
    end
    heuristic=h_j;
    Storeheuristic;
    P[j] = phi * f1[j] + phi * f2[j][best_last_heuristic] + delta * f3[j];
    Apply the selected i;
    repeat
        Update the score of F(i), ∀h ∈ H;
        Select h with the largest F(i), ∀h ∈ H;
        Update the score of P(j), ∀h ∈ H;
        Select h with the largest F(i), ∀h ∈ H;
        Apply the selected i;
        Set S[best] = s[outcome];
        Invoke the acceptance procedure;
    until Stop condition is reached;
end
```

Each individual route consists of multiple pickups and deliveries of patients starting from the source, DE in Figure 6.2. As patients are transported, there are many events for both pickup and delivery. The positive numbers represent pickup, while events in negative numbers denote delivering patients at their destination points.

Heuristics can make modifications on different levels. The first one is the patient to facility assignment level, ensuring that the requirements are properly met. The second modification focuses on the ordering of events in the pickup and delivery process.

6.2.7 Low Level Heuristics

The low level heuristics used in the proposed efficient choice function for the patient transportation problem, include crossover, mutational, ruin-recreate and local search.

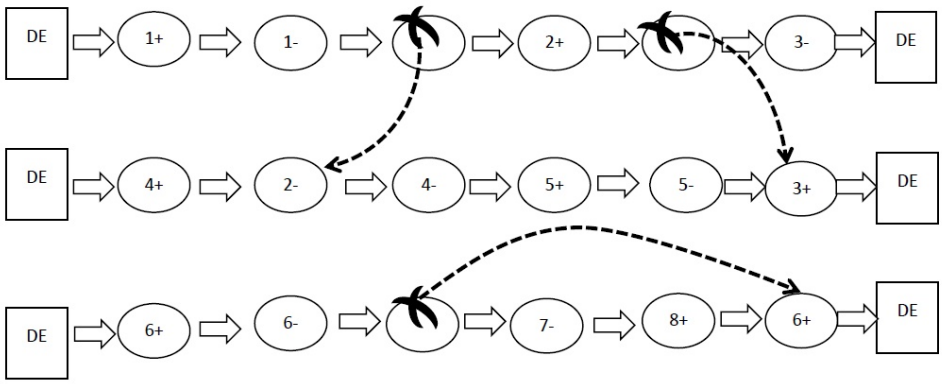

Figure 6.2: Low level heuristics modifications at different levels in routes

6.2.7.1 Crossover Heuristic

The crossover heuristic generates an improved solution from a pair of individuals. Chromosomes from different individuals are combined to form a new genetic offspring. The data from chromosomes based on the single point in the crossover bar is swapped between parents, as shown in Figure 6.3. The new offspring bears the genotype formed from the different organisms of the two parents. However, the crossover heuristic can also be two point, where more than one parents are mated and chromosomes exchanged to produce a better offspring. This helps to perform a rigorous search of a sample space.

There are two points denoting a double-point crossover in Figure 6.4. The performance can be affected if there are many points in the crossover, as expressed by Kaya et al. [147]. The heuristic crossover considers the offspring that is found in the direction of the best fitness parent further away from the poor fitness parent. The heuristic crossover algorithm is presented in Algorithm 14, where a pair of parents is considered for mating to produce

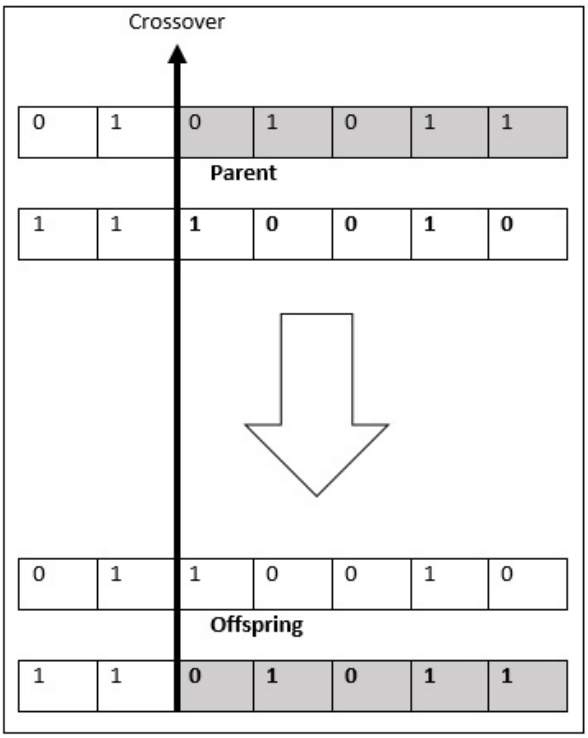

Figure 6.3: Single point crossover heuristic

offspring.

Algorithm 14: Heuristic crossover

$P_1, P_2, S \leftarrow$Initialize Parents and Solutions;
s$\leftarrow$ solutions$\in S$;
Child, C$\leftarrow s$;
while *stop conditions are not met* **do**
  $d(s, s^*)$= direction towards the best solution;
  Combine P_1 and P_2;
  if *Offspring,* $C < P_1$ *and* P_2 **then**
    $C \leftarrow s^*$;
  end
  return C
end

6.2.7.2 Perturbation Heuristic/Mutational

Mutational heuristics are key heuristics used by hyper-heuristics frameworks [148]. They ensure that the search is not trapped into local optima. Mutational heuristics

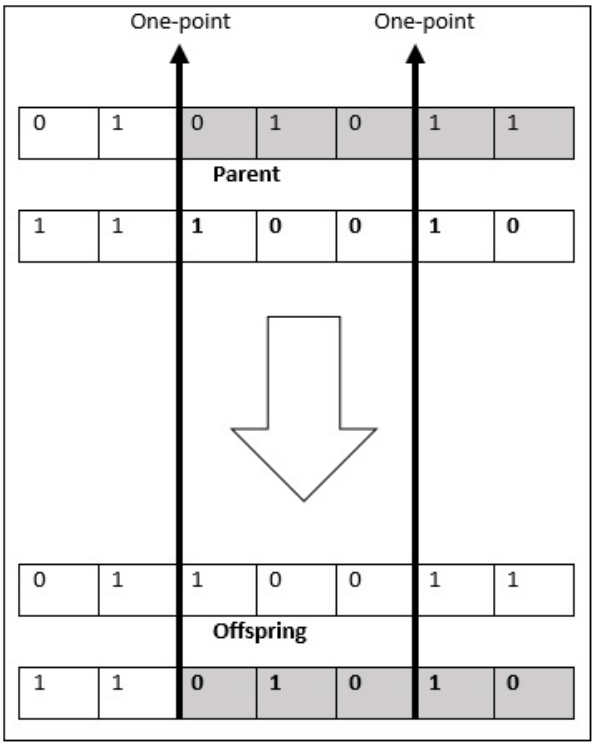

Figure 6.4: Two-point crossover heuristic

provide a synergistic framework for exploring and exploiting the solution sample. The components of different solutions are exchanged between solutions in order to produce a perturbed and improved individual.

Parent	11	9	3	5	2	10	7	12	8	4	6
Offspring 1	11	9	3	5	2	10	8	12	7	4	6
Offspring 2	11	9	7	5	2	10	3	12	8	4	6

Figure 6.5: Example of mutational heuristic

In Figure 6.5, elements are randomly selected and neighbors generated as per the permutations of the chosen genes. The offspring are then generated after evaluation of the neighbors. The method randomly makes a selection of a chromosome gene and swaps

it between chromosomes. The first perturbation heuristics used for implementation in this thesis helps to match the job that was arbitrarily chosen to the position that was also arbitrarily selected. After swapping, a new chromosome is created. There are many other variants of mutational methods that have been widely used and the goal of each is to generate an offspring that is more robust than its genetic parents after combining different genes from solution chromosomes.

Algorithm 15: Perturbation heuristic

Initialize Parents and Solutions;
while *stop conditions are not met* **do**
    Select genes randomly;
    Find permutations of the genes;
    Perform neighbor evaluations;
    Choose the best offspring;
    return Offspring
end

Mutation heuristics overcome premature convergence in hyper-heuristics by extending the search space through mutation of individuals. They also help to intensify and diversify the search process.

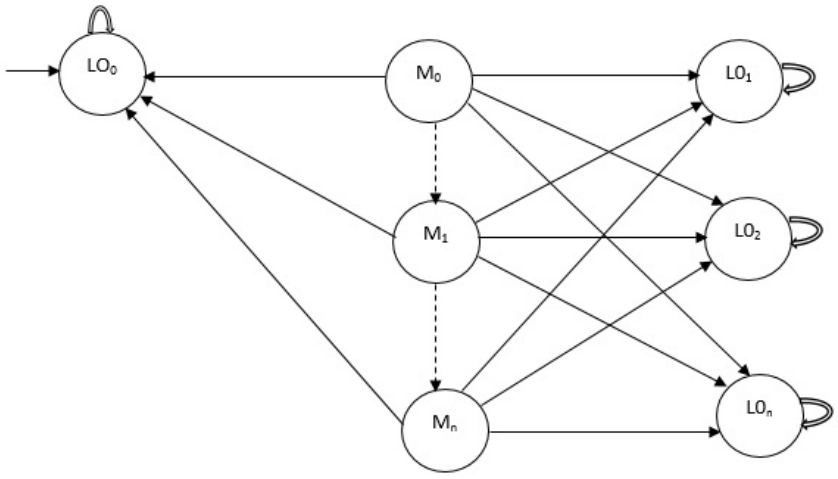

Figure 6.6: Mutational and local search in hyper-heuristics [149]

Figure 6.6 demonstrates how a hyper-heuristic can escape being trapped into local optima. LO and M denote local optimum and mutation respectively. It is shown how, through mutation, the search sample can be extended in order to explore new search areas. This, in turn, improves the quality of the offspring solutions produced. Moreover, combining hill climbing and mutational can also create good improvements in hyper-heuristics.

6.2.7.3 Local Search Heuristic

The low level heuristic is geared towards making incremental improvement changes to the solution. Lourenco et al. [150] presented a framework for a local search heuristic which uses the cost parameter as a base for the solution preference. The lowest cost solution is given precedence over the other solutions. The local search procedure perturbates the original solution, s, to an improved low-cost outcome $s*$ as shown in Figure 6.7. D and C denote density and cost respectively.

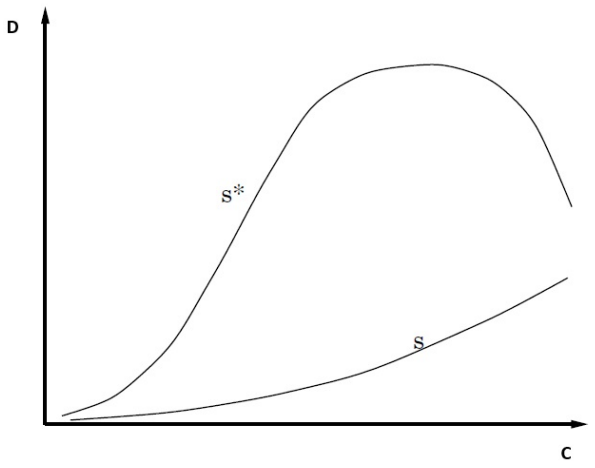

Figure 6.7: Local search improvement

Algorithm 16: Local search improvement

$S \leftarrow$Initialize solutions;
Select initial solution, $s \in S$;
while *criteria is not reached* **do**
    if *f(s')<f(s)* **then**
        Select $s' \in N(s)$;
        $s : -s'$;
    else
        f(s')<f(s);
        Keep s unchanged;
    end
    return s'
end

The local search procedure moves iteratively from solution s to the next solution $s*$ in the solution sample space. Strong individuals are accepted until the algorithm terminates, as presented in Algorithm 16.

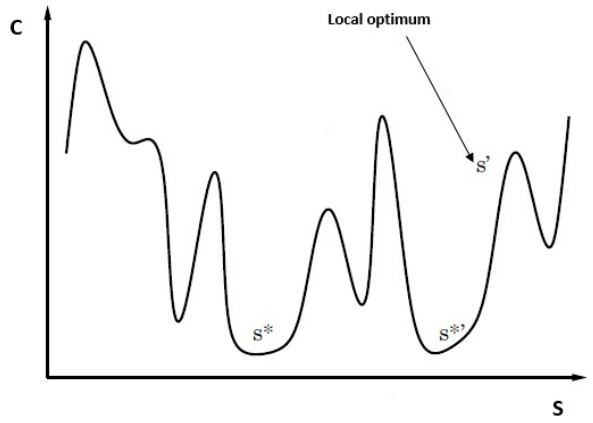

Figure 6.8: Hill climbing improvement heuristic

The local search heuristic is different from a mutational heuristic because of its iterative component, as demonstrated in Figure 6.8. Also, C and S denote cost and sample space respectively. The area around the solution, which is known as the *neighborhood*,

is searched to find out if there is a better individual than the incumbent solution. The better solution is then adopted as the optimal solution. The problem domains within the hyper-heuristics flexible framework are configured for minimization of the objective function to produce the best outcome. The depth and breadth of the search is controlled by the parameters beta, β and alpha α respectively. The best solution, $s*$, is the one with a lower cost, as shown in Figure 6.8.

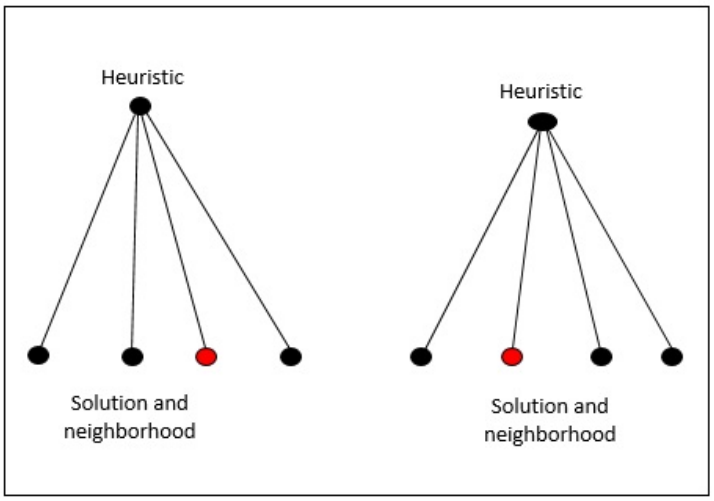

Figure 6.9: Mapping the local search heuristic onto the solutions

Once the starting solution, s, is chosen, a direction is found in the neighborhood towards the minimum of the solution, $f(s) \rightarrow f(s*)$. The solution neighborhood, as demonstrated in Figure 6.9, $N(s), \forall s \in S$ is searched and given the initial solution in *red*. Neighborhood improvement by the local search is conducted within a sample of solutions. An improved perturbed solution generated replaces the incumbent solution, based on a move acceptance criteria.

6.2.7.4 Ruin and Recreate

The aim of a ruin and recreate is to destroy the existing structure of the solution and build a new improved one. As defined by Schrimpf et al. [151], a big part of the solution is ruined or decomposed to pave way for the creation of new structural components of the solution. The decision rule mechanism is used to determine whether the constructed new structure should be accepted or not. Better quality solutions are given precedence as can seen in numerous applications that have implemented a greedy method of accepting

competing perturbed solutions.

Algorithm 17: Ruin and recreate

```
P ←Initialize solutions;
p' ←Select solution to apply ruin procedure;
while stop conditions are not met do
    Determine which mode to use for ruin;
    p^t=ruin[p];
    p*=recreate[p^t];
    if p' > p* then
        Select p';
    end
    return p'
end
```

There are two basic components of the ruin and recreate method. First is disassembling, which takes the existing solution apart [152]. Second is assembling, which pieces together different components of the solution. The procedure for ruin recreate starts with generating solutions and initializing them. The initial permutations are arbitrarily generated. The improvement mechanism is then invoked to mutate the solution and effect changes upon the decision taken by the acceptance mechanism procedure.

The solution is destroyed by breaking it down into components and reconstructing it in order to come up with a better outcome. The ruin recreate heuristic is predefined in hyflex [15]. At every iteration there is a neighborhood to every solution, and the neighborhood contains a solution sample. After the ruin process, the components are reorganized following a stochastic process with other solutions in the neighborhood.

The ruin recreate adds to the quality of the heuristics within the hyper-heuristics flexible framework. Given that an exhaustive neighborhood exploration requires a lot of time, ruin recreate helps the exploration to be focused on the most important parts of individual's neighborhood.

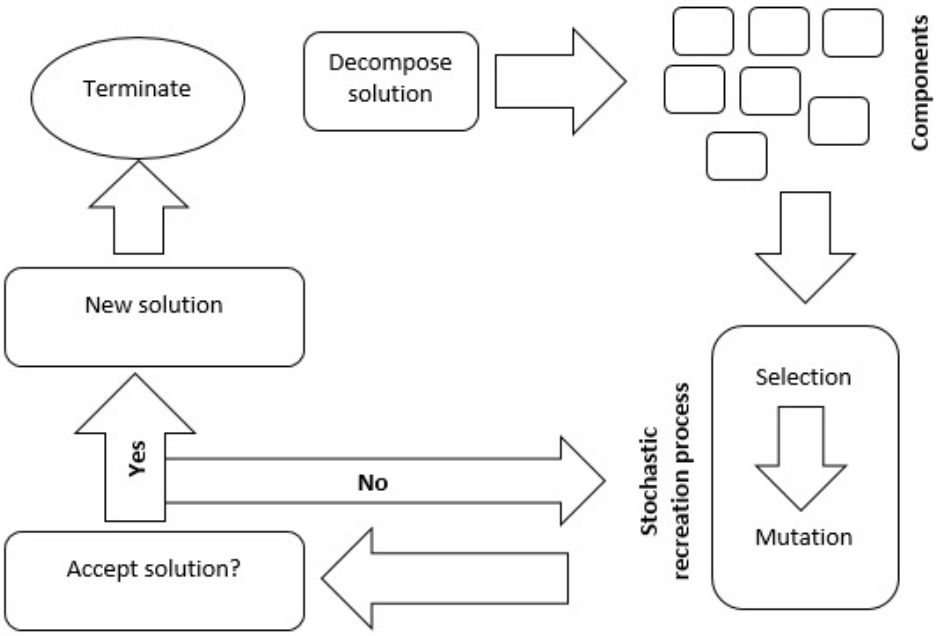

Figure 6.10: Ruin and recreation process

6.3 Results

6.3.1 Efficient Choice Function

One of the problems solved by the proposed hyper-heuristics is the traveling salesman problem, which has already been defined in this chapter. The instances [131] used contain data sets of up to 18 512 cities. The weighted sum translates into fitness of each best outcome, whose data is plotted from different runs to demonstrate the performance of different hyper-heuristics.

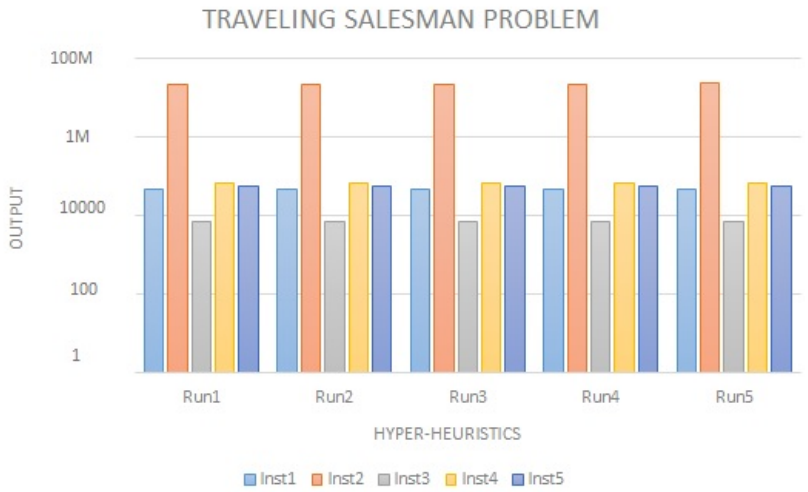

Figure 6.11: Performance of the proposed hyper-heuristics on a traveling salesman problem

Figure 6.11 demonstrates the average performance of the hyper-heuristics with a proposed choice function. As can be observed, the second heuristic maintains consistency in performing better than other methods. The orange bar in the chart corresponds to the best outcome generated by a hyper-heuristic and it is this hyper-heuristic in every set of heuristics that emerges with better performance after applying a single-point search mechanism. The outcome is also attributed to efficient control of parameters to avoid a search falling into local optima in addition to balancing search intensity and diversity in the search sample.

Figure 6.12: Comparison between the existing hyper-heuristics in blue and the proposed hyper-heuristics in orange

Figure 6.12 shows the comparison between the existing method and hyper-heuristics with the proposed efficient choice function. At first the existing hyper-heuristic show better performance. This is because the iterations for improving the individuals are still low at that time , which affects search learning. However, the performance pattern changes after the second instance, showing a leap in performance from the proposed hyper-heuristics, and another leap is observed in the fifth run.

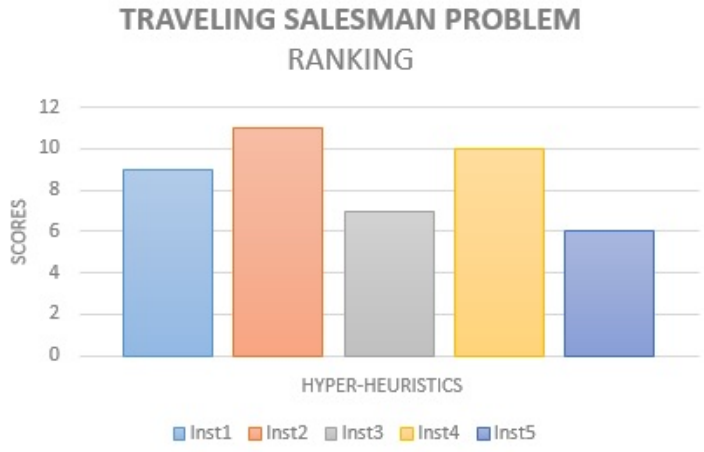

Figure 6.13: Traveling salesman problem weighted score

The ranks for each heuristic are shown in Figure 6.13. A second heuristic from the left obtains the best score due to a sustained good performance exhibited by the heuristic in the course of the runs. At the start, some initial individuals are weak in the search, which affects the initial search outcome.

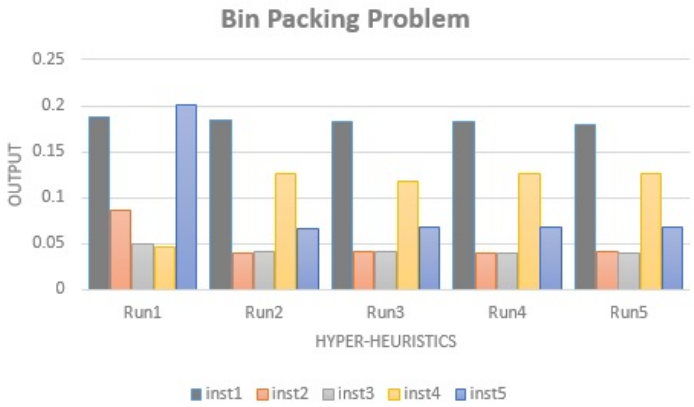

Figure 6.14: Performance of the proposed hyper-heuristics on a bin packing problem

A bin packing problem is another problem, which was used to test the methods. A set of items must be put into a finite-sized bin. All items are packed subject to the few available bins. If some items are not packed, then the solution becomes partial, as explained by Soubeiga [153]. Typically, the solution begins as partial given that there are no packed items at the start. The instances used a range from 100 to 1000 capacity and the pieces start from 160 to 5000. The first heuristic, from left in Figure 6.14, performs well across different runs. The improving technique increases the efficiency of the search when it is combined with the perturbative method on a bin packing problem.

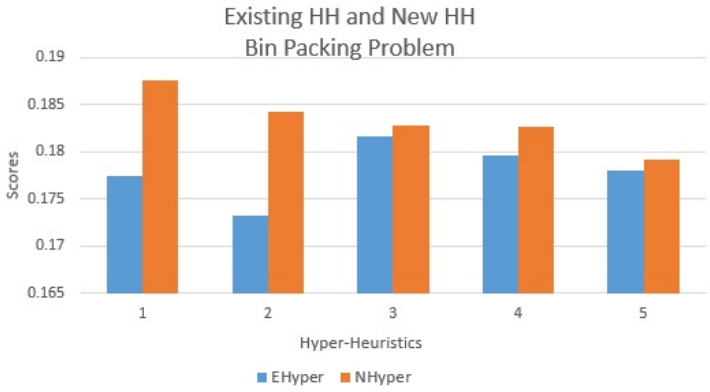

Figure 6.15: Comparison between different methods for a bin packing problem

Figure 6.15 compares the proposed hyper-heuristics with the existing method. The proposed method performs sustainably well on all instances due to the efficiency of an efficient choice function. The packing problem requires continuous improvement, instead of invoking reinitialization all the time.

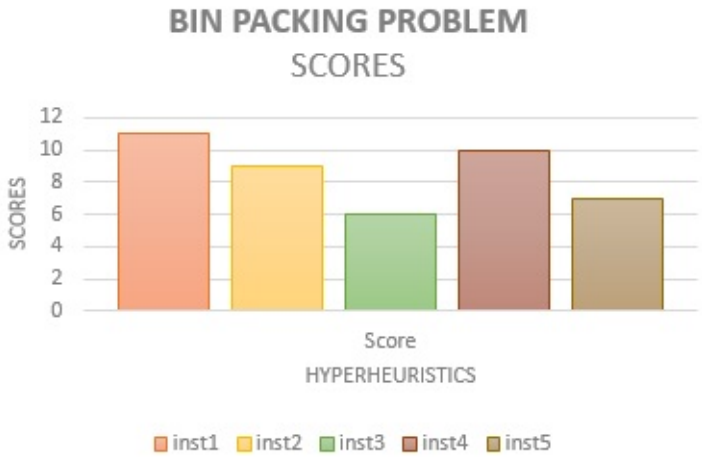

Figure 6.16: Bin packing problem weighted score

Figure 6.16 presents the weighted score of each heuristic when applied to a bin packing problem. In addition to a local search, a number of heuristics such as mutation, crossover and ruin recreate are applied, which prevent premature convergence.

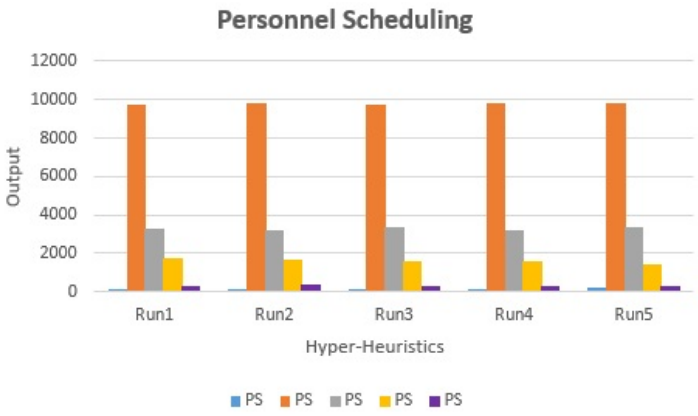

Figure 6.17: Performance of the proposed hyper-heuristics on a personnel scheduling problem

Personnel scheduling deals with scheduling tasks and minimizing the time it takes to execute those tasks. The proposed method generates good outcomes when applied to this problem. The dismal performance noticed in Figure 6.17 on some heuristics is in part due to the fact that those heuristics are completely not suited to the problem, while on the other instances better performance is observed.

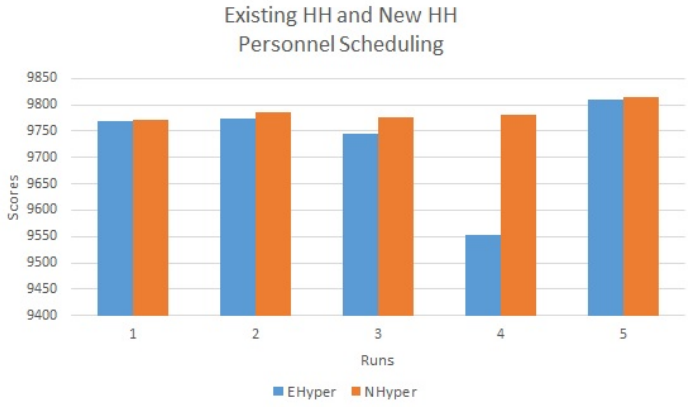

Figure 6.18: Comparison between different methods for a personnel scheduling problem

In Figure 6.18, the proposed method outperforms the existing hyper-heuristics, especially after the improvement of individuals when the low level heuristics are employed.

Initially, there is an observable tie in performance, and subsequently a small difference, until a leap is observed after the third instance in the proposed approach. The existing method shows another burst of good performance in the fifth instance, and the proposed method consistently yields good results.

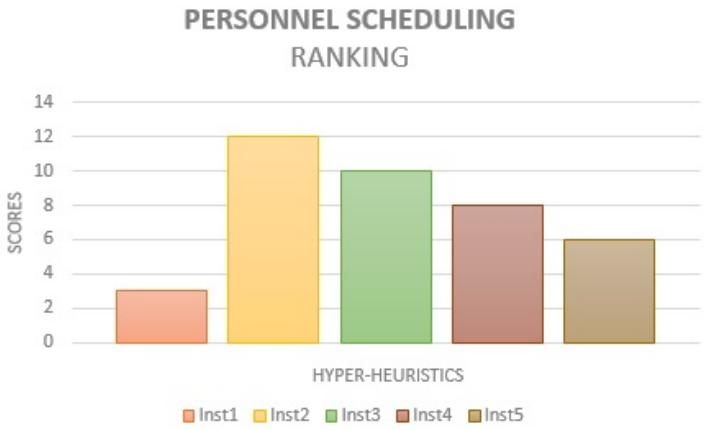

Figure 6.19: Personnel scheduling problem score

Figure 6.19 demonstrates the ranking of the different heuristics when applied to a personnel scheduling problem. Using instances from standard datasets by Curtois [136], a diverse set of staff is used from 12 to 51 with different shifts and multiple days from 26 to 42. The graph encapsulates the average ranking of each heuristic, based on the performance generated.

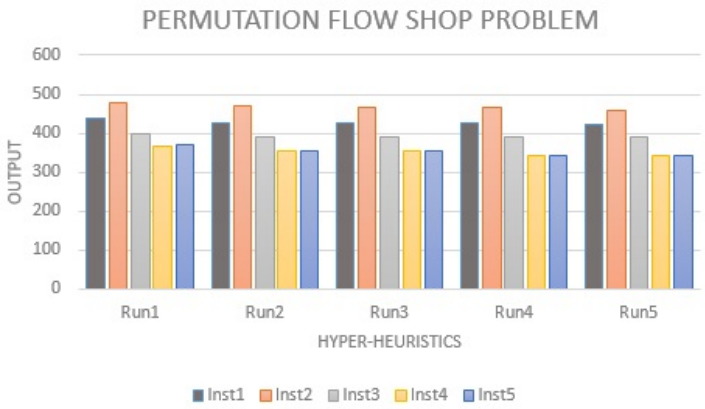

Figure 6.20: Performance of the proposed hyper-heuristic on the permutation flow shop

A permutation flow shop problem defines the task processing order successively, while considering a specific machine's number available. There is a procedure or an order on which the processing is based on and one task is handled at a time so that the procedure is maintained throughout the iterations. Moreover, the machine cannot remain unselected once it is ready. The standard datasets by Taillard [154] were used for these experiments.

The procedure by Nawaz et al. [155] was adopted for the initialization of the solutions and other elements. The jobs to be handled ranged from 100 to 500 and machines ranged from 10 to 20 for different instances. The results shown in Figure 6.20 demonstrate good competition from different heuristics. The second heuristic from the left across all instances shows sustained better performance than the rest of the algorithms.

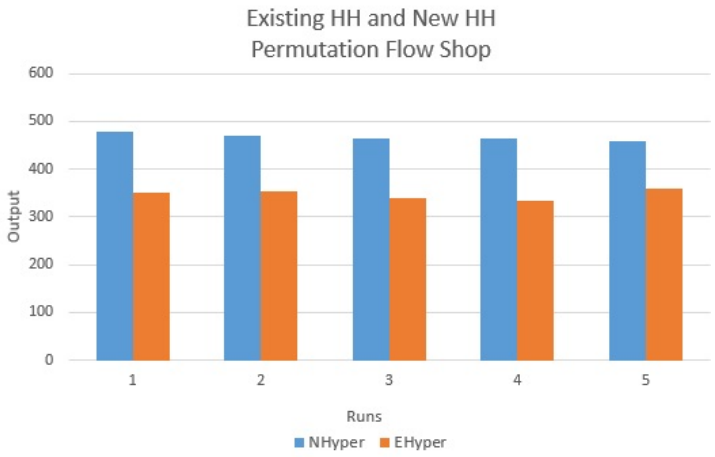

Figure 6.21: Comparisons between different methods for permutation flow shop

The proposed method, as shown in Figure 6.21 with blue bars, outperforms the existing method in orange bars in the chart. This is due to the good scores generated based on an efficient choice function.

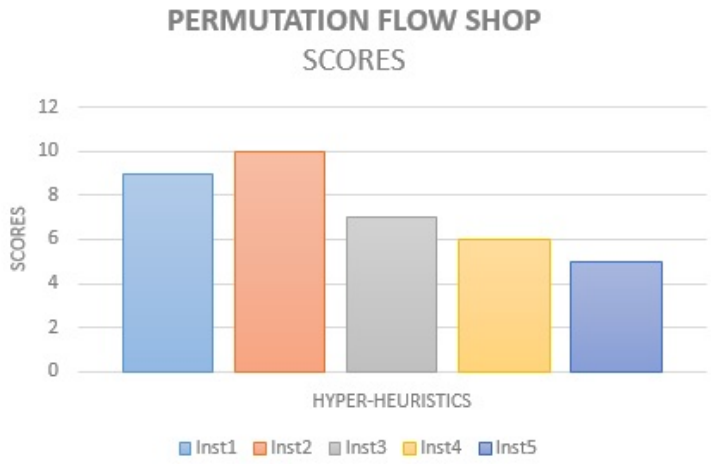

Figure 6.22: Permutation flow shop ranking

Figure 6.22 shows the ranking of each heuristic with regard to the performance shown by each method. It is the second method that performs better. The goal of this problem is to reduce the length of the period required to process the number of scheduled tasks.

The second heuristic is ranked highest and shows faster convergence.

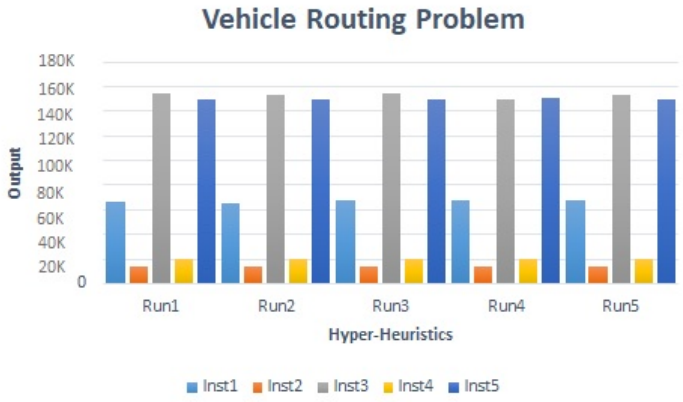

Figure 6.23: Performance of the proposed hyper-heuristic on a vehicle routing problem

A vehicle routing problem is concerned with being able to visit specific locations in a finite period using a minimum number of vehicles available. The general objective is to reduce the required vehicles and ensure a reduction in the cost distance. Vehicles from 3 to 100 and distance from 1061 to 63 373 are considered over a number of data instances, as defined in [148,156]. In the third instance, a leap in performance is observed in Figure 6.23, which is sustained throughout the iterations. This is also demonstrated in Figure 6.24, where performance by the proposed method improves with time during the search until it is sustained.

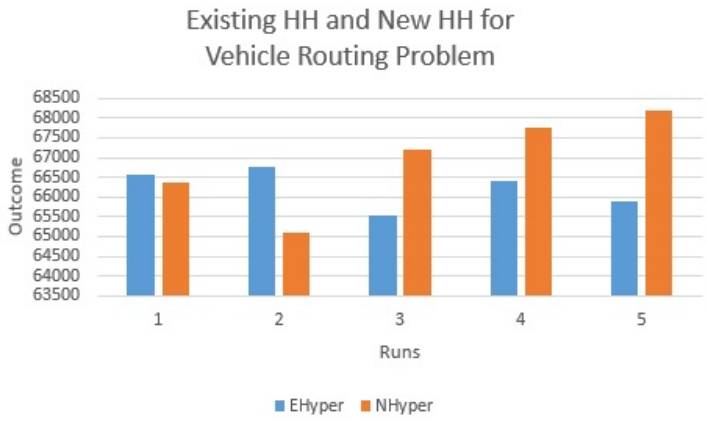

Figure 6.24: Comparison between different methods for vehicle routing problem

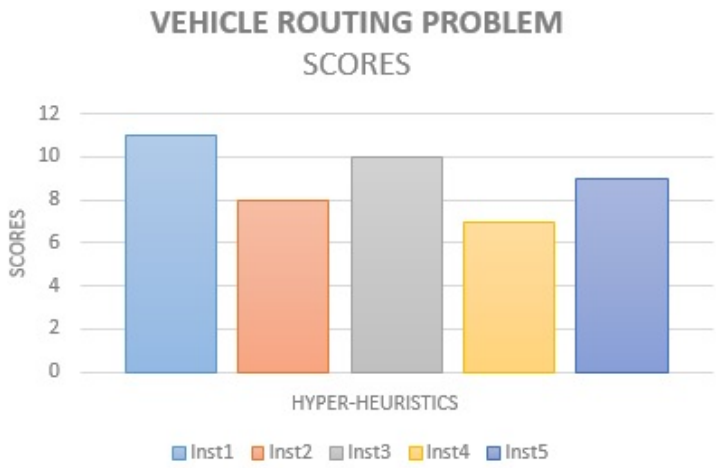

Figure 6.25: Vehicle routing problem weighted score

Figure 6.25 shows a heuristic ranking after solving a vehicle routing problem. The best scored method is shown by the light blue bar in the chart.

6.3.2 Dynamic Selection Hyper-heuristics

The proposed hyper-heuristics was also applied to a real-life patient transportation problem (PTP) from datasets in [145]. An Intel Pentium 3.0GHz Dual Core Processor

and 4GB memory were used for this experiment. The maximum travel time is set for each vehicle, including the disinfenction time and the vehicle capacity. The learning components are incorporated within the heuristic selection mechanism. One of the goals of this approach is to reduce the time it takes for the service to be rendered.

The storage of weak individuals for recall by the hyper-heuristics and the continuous performance check by the alternative heuristic, as proposed in the efficient choice function, help to yield good performance when applied to hospital requests and service delivery.

ANOVA

Solution

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	2275.504	1	2275.504	44.261	.000
Within Groups	2981.808	58	51.410		
Total	5257.313	59			

Figure 6.26: Analysis of variance

	N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
					Lower Bound	Upper Bound		
Existing HH	30	53.9667	6.46734	1.18077	51.5517	56.3816	39.00	67.00
New HH	30	66.2833	7.80990	1.42589	63.3671	69.1996	50.00	80.00
Total	60	60.1250	9.43965	1.21865	57.6865	62.5635	39.00	80.00

Figure 6.27: Statistics descriptions

The tools for statistical analysis used are an analysis of variance (ANOVA) test and one sample test, as shown in Figures 6.26 and 6.27. If the p-value is less than 0.05, the null hypothesis is rejected and if p-value is greater than 0.05, the null hypothesis is accepted. The proposed approach generates higher scores for the problem, which implies a stronger performance, as shown in Table 6.2. The existing hyper-heuristics and the

proposed hyper-heuristics in Table 6.2 are denoted by E-hyper-heuristic and P-hyper-heuristic respectively. Convergence improves after a slow performance between 23 and 25 instances as the weighted scores are calculated by an efficient choice function in a changing search environment.

Table 6.2: Hyper-heuristics on dynamic transportation problem

		Performance on the DTP					
Run	E-hyper-heuristics	Distance	P-hyper-heuristics	Run	E-hyper-heuristics	Distance	P-hyper-heuristics
1	52	44056	65	16	56	43116	75
2	47	43804	71	17	47	43116	60
3	55	44457	85	18	44	40827	78
4	59	44772	80	19	61	41157	65
5	59	45069	62	20	58	42312	80
6	51	43964	77	21	49	43153	70
7	55	43699	75	22	51	44748	75
8	67	43572	80	23	43	44058	57
9	58	44199	88	24	60	44683	60
10	57	44373	78	25	57	43215	58
11	61	44916	65	26	58	42582	68
12	46	44367	72	27	58	44135	78
13	49	44471	70	28	55	43706	70
14	51	44201	85	29	58	41122	83
15	57	43166	82	30	63	40418	76

In the application of the efficient choice function to a dynamic selection hyper-heuristic using a dynamic Gaussian process regression, implementations were conducted using the toolbox of Gaussian processes for machine learning, as defined by Rasmussen and Nickisch [157]. The instances used for this experiment are obtained in [158]. The search parameters of the covariance function are set to $\ell = 2, \sigma_f^2 = 1$ and the local search improvement is also incorporated to prevent premature convergence. In addition, the comparison techniques such as genetic algorithm and simulated annealing are used. In terms of estimation deviations, a formula [159] is invoked to determine the deviations while observing the distinct differences caused by the proposed method. The deviation is defined by the actual predictive outcome minus expected outcome and is divided by the expected outcome, which is then converted to percentage. The predictive and expected outcomes are denoted by y and $y*$ respectively. This is expressed as follows:

$$deviation(\%) = (\frac{\hat{y} * - y*}{y*})100 \tag{6.26}$$

The difference between the estimate and optimal solution is calculated in percentage as 16.64%, which gives a contrast and shows good performance from the estimation. This is followed by time for Gaussian process regression at 4.6402m and the existing method at 8.5301m in a distance of 253.000 traveled by a salesman under a changing environment. The local search is used to improve performance through making refinements. The first path is mapped out by a 2-opt technique, which is used to perform edge swapping. The method constructs points across the route that show where to start and the places to visit along the path. Strict observance of the loop is ensured so that every vertex is visited before returning to the area of departure. The interchange method carves out a new path altogether over the course of swapping vertices.

(I) Algorithm	(J) Algorithm	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
DGPR	GPR	8.01310*	.74117	.000	6.5560	9.4702
	GA	9.48510*	.74117	.000	8.0280	10.9422
	MA	9.03610*	.74117	.000	7.5790	10.4932
GPR	DGPR	-8.01310*	.74117	.000	-9.4702	-6.5560
	GA	1.47200*	.74117	.048	.0149	2.9291
	MA	1.02300	.74117	.168	-.4341	2.4801
GA	DGPR	-9.48510*	.74117	.000	-10.9422	-8.0280
	GPR	-1.47200*	.74117	.048	-2.9291	-.0149
	MA	-.44900	.74117	.545	-1.9061	1.0081
MA	DGPR	-9.03610*	.74117	.000	-10.4932	-7.5790
	GPR	-1.02300	.74117	.168	-2.4801	.4341
	GA	.44900	.74117	.545	-1.0081	1.9061

*. The mean difference is significant at the 0.05 level.

Figure 6.28: This shows multiple comparisons between different algorithms on 100 instances

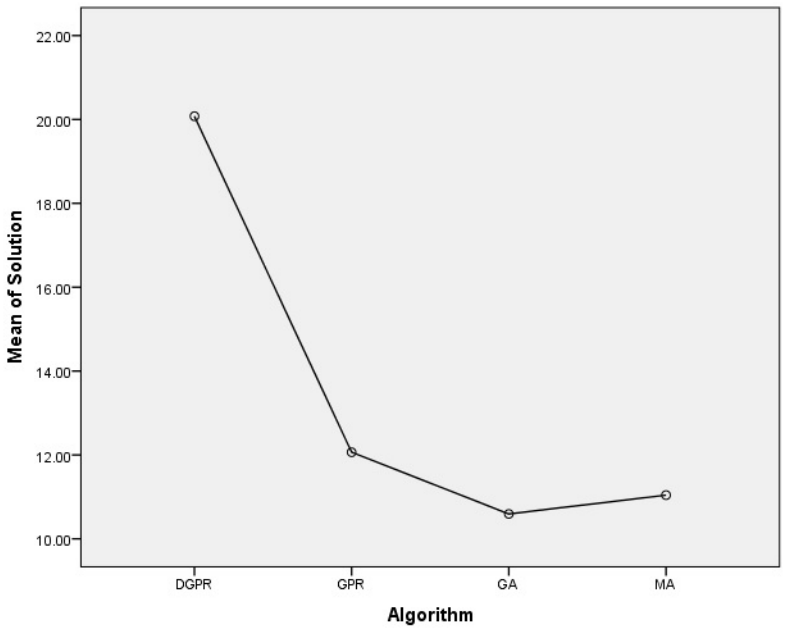

Figure 6.29: Means plots of different algorithms on 100 instances

Test of Homogeneity of Variances

Solution

Levene Statistic	df1	df2	Sig.
16.391	3	396	.000

ANOVA

Solution

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	5981.073	3	1993.691	72.586	.000
Within Groups	10876.778	396	27.467		
Total	16857.851	399			

Figure 6.30: Analysis of variance and test of homogeneity

Descriptives

Solution

	N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean		Minimum	Maximum
					Lower Bound	Upper Bound		
DGPR	100	20.0781	7.39614	.73961	18.6105	21.5457	3.00	38.60
GPR	100	12.0650	4.43939	.44394	11.1841	12.9459	.00	20.50
GA	100	10.5930	4.32193	.43219	9.7354	11.4506	.00	18.00
MA	100	11.0420	4.09590	.40959	10.2293	11.8547	.00	18.00
Total	400	13.4445	6.50002	.32500	12.8056	14.0835	.00	38.60

Figure 6.31: Statistical descriptives on 100 instances

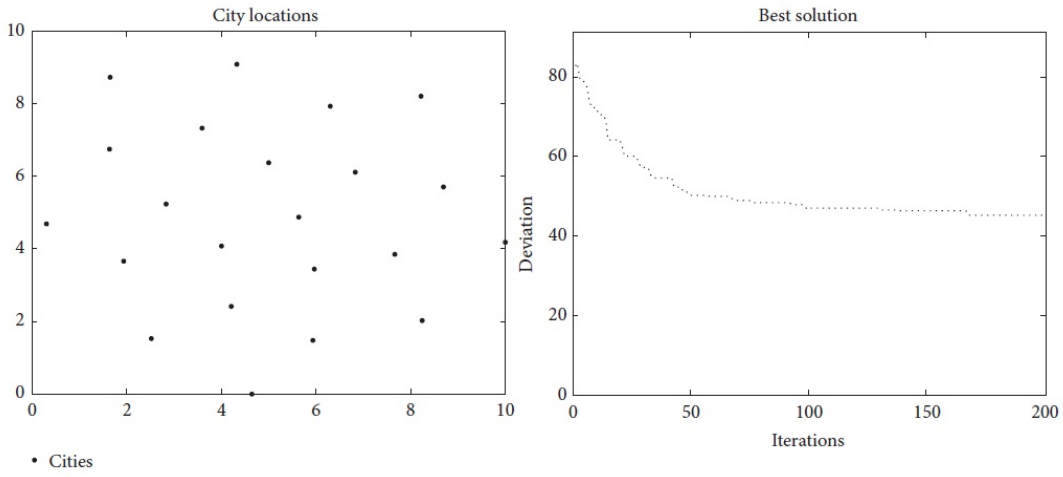

Figure 6.32: The best solution obtained from the city sample

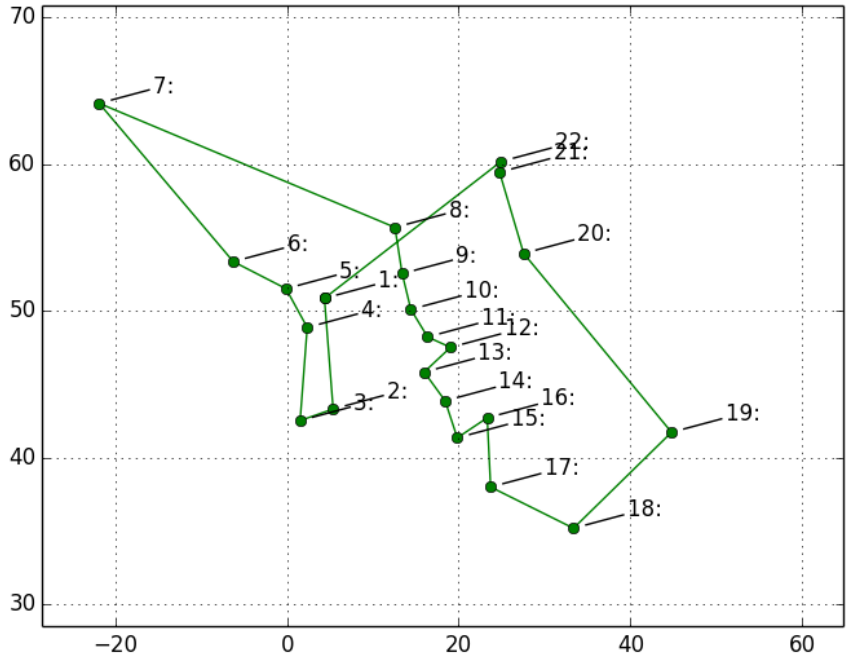

Figure 6.33: This shows the best path obtained in 100 runs

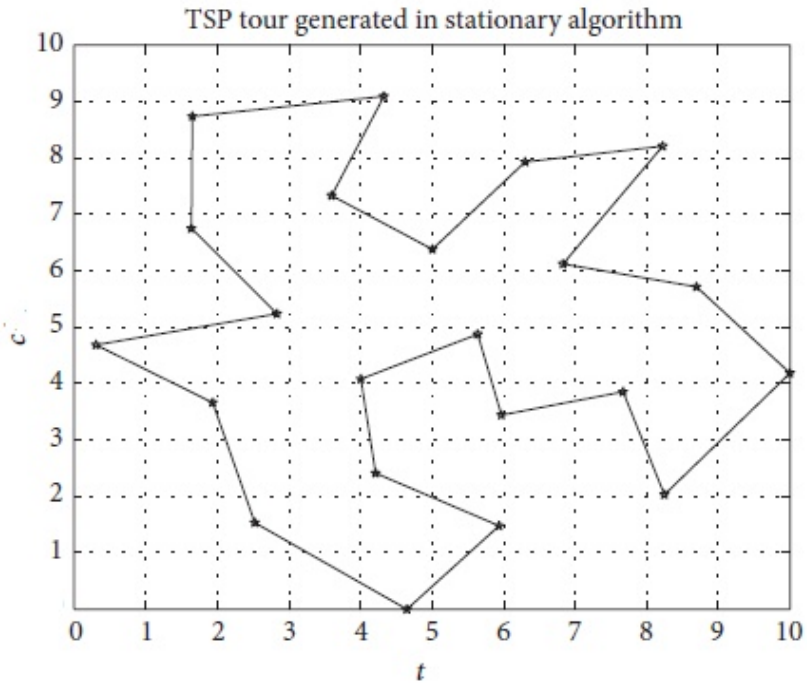

Figure 6.34: Tour completed by the traveling salesman

The statistical analysis tools used in this thesis are an analysis of variance (ANOVA) test and one sample test, which generated results in Figures 6.28, 6.29, 6.30 and 6.31 on 100 instances. The mean difference is shown in Figure 6.29 for the proposed Gaussian process regression for hyper-heuristics against memetic algorithm, genetic algorithm and existing Gaussian process regression. Figure 6.30 shows an analysis of variance and a homogeneity test.

The null hypothesis can be accepted or rejected, based on statistical observations on data generated from 100 instances and the threshold value, p-value. If the p-value is less than 0.05, the null hypothesis is rejected and if p-value is greater than 0.05, the null hypothesis is accepted. Figure 6.32 shows the best solution obtained from the sample. At the end of the run the best solution from a sample of cities is obtained in Figure 6.32 and an optimal route is shown in Figure 6.33. A traveling salesman problem tour is generated in Figure 6.34.

6.3.3 Spy Search Method

The proposed spy search method for memetic search was implemented on a set of dynamic problems and compared with the existing methods. There are different severities

that are used to determine the rate of change in the environment so as to monitor the strength of each method under similar conditions, and the parameter for calibrating speed is δ, which is set to 10, 20, 60 and 100 for oneMax, RoyalRoad, Deceptive and Plateau respectively and the severity change is respectively set at 0.1, 0.3, 0.5 and 0.7.

In the experiments, performance comparisons are made between the proposed method and other known existing methods. All methods are configured with similar settings to determine their comparative performance. The population is set to 100, 0.2 and 0.05 for crossover and mutation respectively.

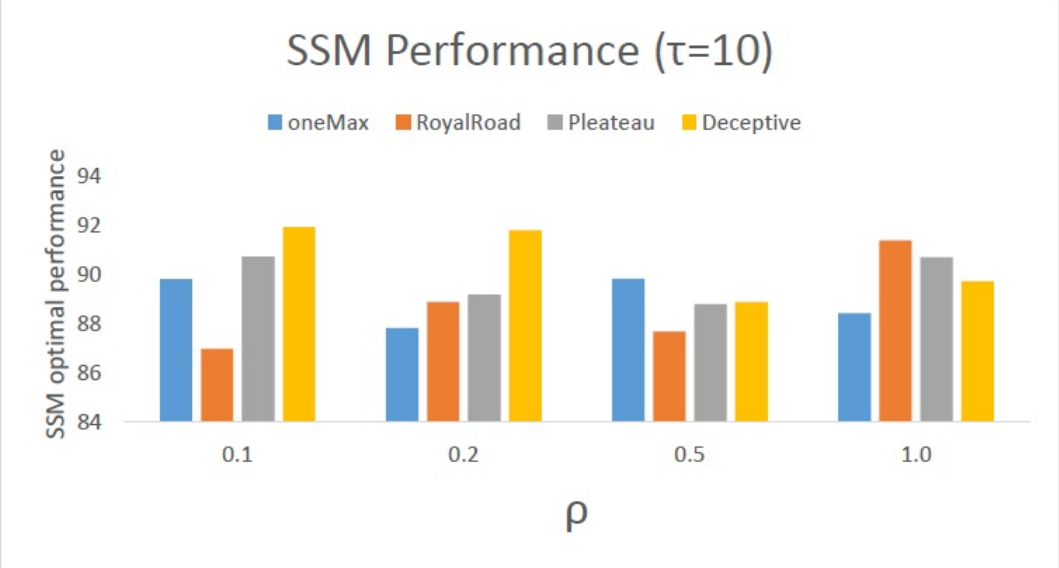

Figure 6.35: SSM optimal performance when $\tau = 10$ for all the problems

The evaluation rate is also set to 0.2. In SSM α, β, Δ and θ are set with 1.0, 1.2, 0.5 and 0.2 respectively. The execution of 24 individual runs was done with randomly similar seeds defined per generation with multiple landscape changes allowed. The best performance outputs were then recorded for each method, as shown in Tables 6.3, 6.4, 6.5 and 6.6.

From the experiments, a spy search method performs better on all the dynamic optimization problems that were generated by the problem generator. The strength of a spy search method is induced by the components that are subsumed in the method to increase diversity and overcome the continuous changes of the search environment. In addition, an increase in performance is observed when the changes occur and when they

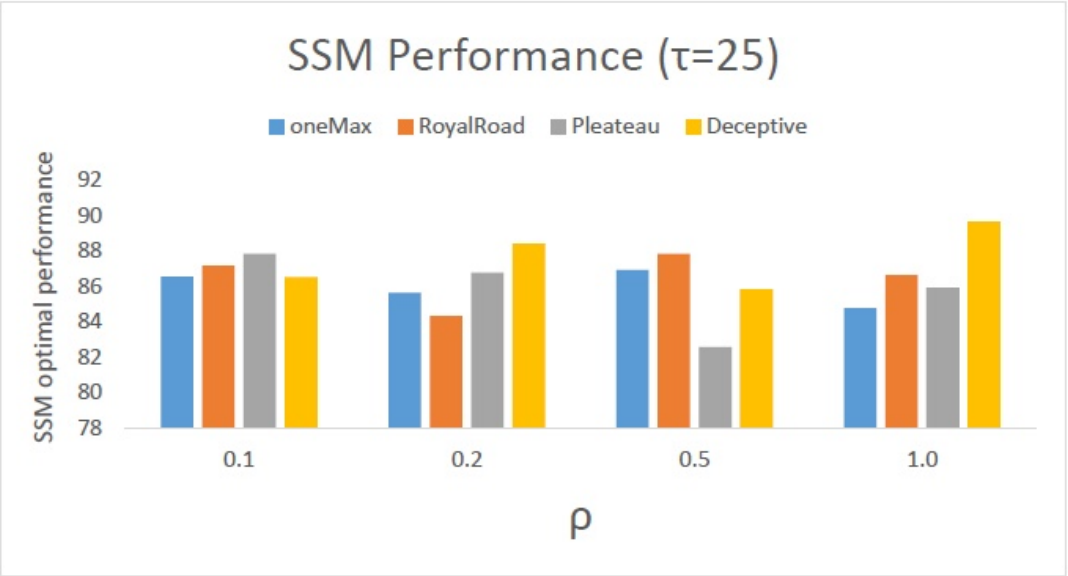

Figure 6.36: SSM optimal performance when $\tau = 25$ for all the problems

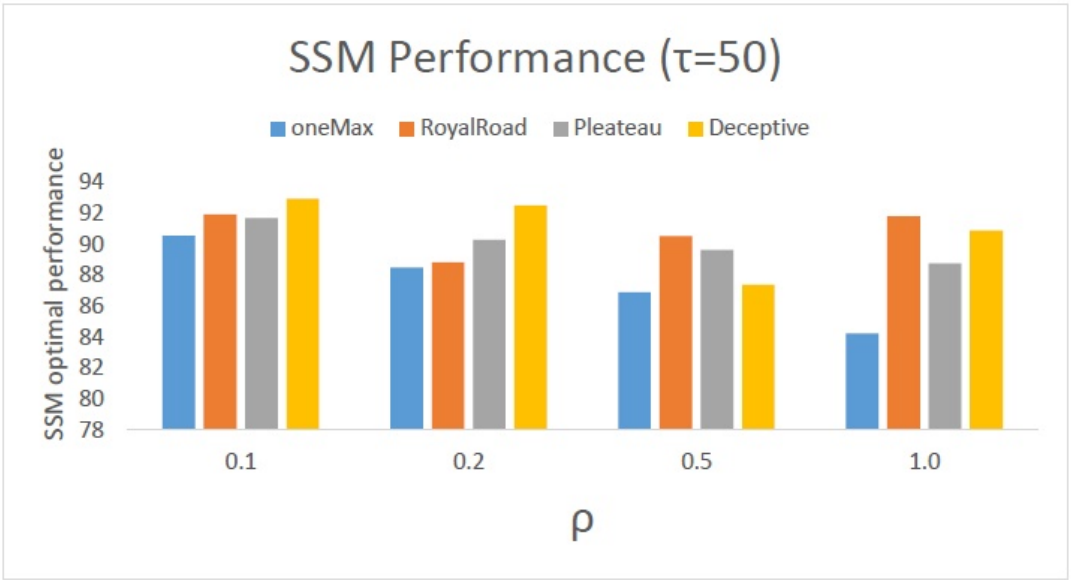

Figure 6.37: SSM optimal performance when $\tau = 50$ for all the problems

intensify, based on the variables set for the parameters. Moreover, a spy search method consistently shows bursts of strong performance at selective intervals of a dynamic search. Figures 6.35 and 6.36 show performance when τ is at 10 and 25 respectively. This is when there are not a lot of changes in the search environment, thus keeping diversity at low levels.

In the measurement of performance and in order to draw a direct contrast with ex-

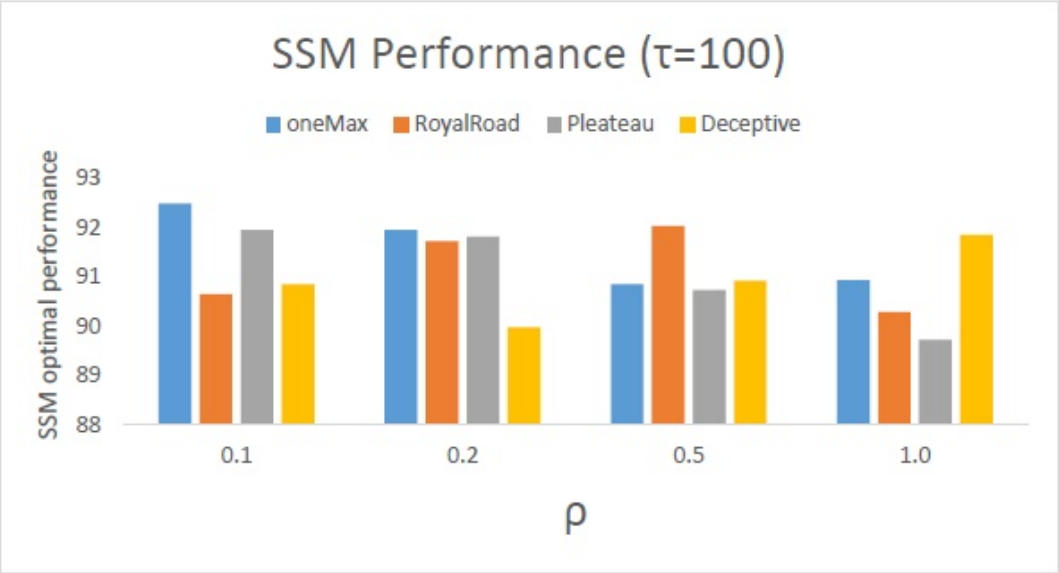

Figure 6.38: SSM optimal performance when $\tau = 100$ for all the problems

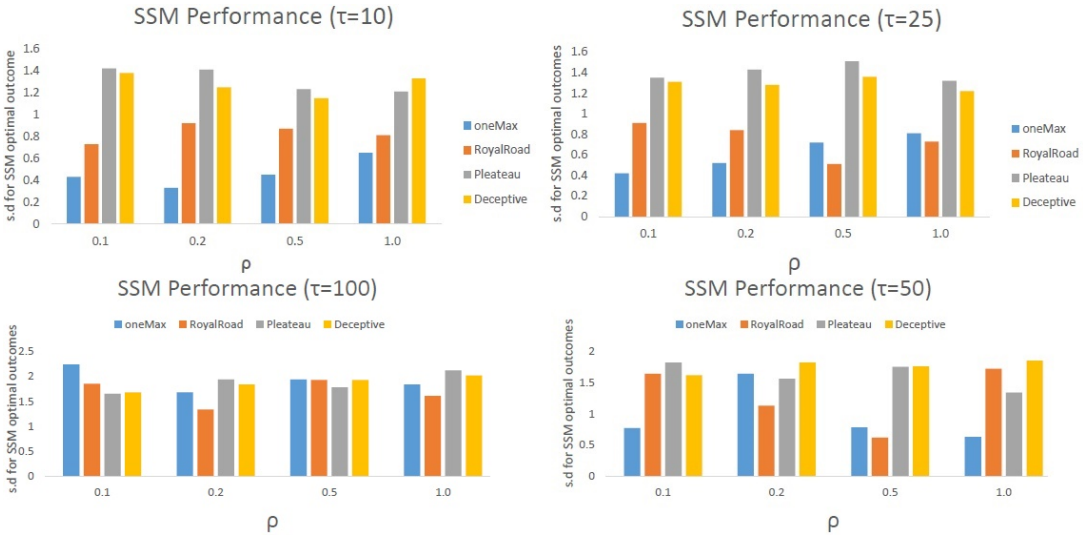

Figure 6.39: SSM s.d for optimal performance for all the problems

isting methods, 24 runs are considered for the algorithms and the fitness over the runs is calculated [112]. The performance of a spy search method is better when compared with other existing methods, as shown in Tables 6.3, 6.4, 6.5 and 6.6, due to its diversity and information collected to guide the search. The diversity of the population in each generation is defined based on the size of the population and the distance between individuals in the search [112]. The size of τ alternates between 10, 25, 50 and 100 so as to affect diversity accordingly. The generations are set to 100 for individual runs and

it is noticeable that a spy search method is stronger although it struggles to compete in a few multiple runs, especially as observed in Table 6.6, because of its rigorous search process that includes information collection and analysis. It can be noticed that when τ is increased, there is an increase in the SSM output, although a decrease is detected in some instances in the course of the run. The decrease is because of slow changes in the environment. The opposite is when the changes are happening rapidly as balanced by τ, and there is quick convergence resulting from search exploitation by a local search.

Figure 6.38 shows a high rate of performance due to a strong diversity created by a hybrid of immigrants method, hypermutation and crowding and fitness sharing. But when the environment begins to change, convergence slows down, as seen in Figure 6.39, when the $\tau = 25$. As the level of severity increases, the diversity methods are invoked to counter the volatility of the changing search environment which, in turn, creates an increase in performance, as noticed in Figures 6.37 and 6.38 when τ changes from 50 to 100. OneMax performs relatively well when τ keeps increasing up to 100.

Table 6.3: Comparison results on dynamic problems ($\tau = 10$)

		Performance on oneMax problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
10	0.1	81.32±0.03	80.21±0.12	81.28±0.14	80.62 ±0.10	**89.8±0.43**	82.16 ±0.18	78.93±0.15	81.12 ±0.17	79.41±0.11	80.50±0.18
10	0.2	84.67±0.06	81.51±0.22	80.58±0.24	85.82 ±0.26	**87.82±0.33**	83.26 ±0.20	79.84±0.25	82.22 ±0.19	82.51±0.21	79.35±0.22
10	0.5	83.52±0.05	86.21±0.08	84.28±0.04	88.62 ±0.19	**89.82±0.45**	83.56 ±0.38	88.93±0.45	86.12 ±0.27	87.41±0.31	85.81±0.68
10	1.0	82.25±0.55	83.51±0.48	84.28±0.34	85.98 ±0.69	**88.42±0.65**	85.57 ±0.78	84.94±0.75	86.12 ±0.47	84.41±0.71	87.86±0.88
		Performance on RoyalRoad problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
10	0.1	83.43±0.83	78.58±0.62	85.11±0.45	85.68 ±0.81	**86.98 ±0.73**	79.73 ±0.38	78.93±0.95	82.13 ±0.77	74.91±0.78	83.52±0.18
10	0.2	84.43±0.67	85.47±0.68	83.22±0.59	78.64 ±0.92	**88.87 ±0.92**	76.70 ±0.28	76.32±0.14	82.43 ±0.74	84.19±0.68	85.75±0.39
10	0.5	88.47±0.57	82.71±1.37	63.48±1.45	68.82 ±0.87	**87.67 ±0.87**	79.98 ±1.13	86.75±0.56	87.78 ±0.68	86.21±0.82	84.26±1.23
10	1.0	90.21±1.78	86.84±0.67	86.52±0.92	83.24 ±1.83	**91.37 ±0.81**	89.27 ±0.89	88.69±0.94	90.45 ±1.58	87.57±0.71	86.91±1.78
		Performance on Plateau problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
10	0.1	89.31±0.28	88.76±0.63	87.75±0.39	87.52 ±0.87	**90.72 ±1.42**	86.39 ±0.69	89.62±0.31	89.39 ±1.43	87.72±1.82	86.59±0.85
10	0.2	87.49±0.57	85.52±0.91	86.72±0.42	67.75 ±0.91	**89.18 ±1.41**	88.92 ±1.41	72.82±0.58	71.28 ±1.29	85.08±1.09	85.91±0.39
10	0.5	88.61±0.28	87.75±1.21	85.48±1.03	83.72 ±0.76	**88.79 ±1.23**	77.92 ±0.45	76.87±0.33	81.55 ±1.12	80.29±0.21	82.89±0.82
10	1.0	89.78±0.66	87.28±0.54	76.82±0.88	85.11 ±0.62	**90.68 ±1.21**	88.71 ±0.66	77.95±0.67	87.92 ±0.23	78.19±0.47	88.71±0.49
		Performance on Deceptive problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
10	0.1	91.85±1.54	89.92±0.81	82.59±0.71	89.35 ±0.76	**91.92 ±1.38**	82.87 ±0.47	90.74±0.82	89.27 ±1.62	83.76±0.18	82.57±0.91
10	0.2	90.64±0.76	89.39±0.98	88.54±0.92	89.73 ±1.46	**91.79 ±1.25**	90.72 ±0.82	71.38±0.19	72.85 ±0.66	87.88±0.52	86.28±0.62
10	0.5	87.78±0.65	88.43±1.06	87.72±0.76	86.39 ±0.46	**88.87 ±1.15**	86.81 ±0.71	82.62±0.81	87.92 ±1.87	86.72±0.76	85.91±0.29
10	1.0	88.63±0.71	84.71±1.23	86.94±1.17	85.26 ±1.08	**89.72 ±1.33**	87.29 ±0.88	83.79±0.54	79.88 ±1.59	84.48±0.69	79.39±0.75

Table 6.4: Comparison results on dynamic problems ($\tau = 25$)

	Performance on oneMax problem										
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
25	0.1	78.61±0.13	75.62±0.67	74.49±0.31	82.71 ±0.62	**86.54±0.42**	83.67 ±0.22	83.24±0.54	85.33 ±0.51	76.27±0.27	75.48±0.44
25	0.2	82.31±0.81	80.38±0.17	79.91±0.58	75.77 ±0.17	**85.63±0.52**	84.87 ±0.52	83.92±0.65	76.29 ±0.22	67.81±0.66	77.82±0.91
25	0.5	78.78±0.06	77.38±0.02	78.92±0.03	82.26 ±0.25	**86.93±0.72**	85.71 ±0.56	87.88±0.65	84.65 ±0.39	83.65±0.47	85.92±0.59
25	1.0	81.73±0.72	80.28±0.74	79.91±0.52	65.82 ±0.87	**84.76±0.81**	81.28 ±0.63	83.82±0.88	79.43 ±0.28	80.54±0.87	77.87±0.68
	Performance on RoyalRoad problem										
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
25	0.1	84.37±0.71	78.58±1.28	85.11±1.95	86.44 ±0.89	**87.17 ±0.91**	75.81 ±0.21	73.65±0.34	82.48 ±0.67	83.69±0.52	82.74±0.03
25	0.2	83.77±0.82	82.18±0.43	80.42±0.48	71.28 ±0.13	**84.32 ±0.84**	84.19 ±0.41	75.78±0.52	74.37 ±0.61	83.28±0.42	82.21±0.12
25	0.5	85.17±0.12	81.97±1.72	76.22±1.95	84.24 ±0.53	**87.83 ±0.51**	86.72 ±1.48	86.75±0.17	85.62 ±0.45	84.38±0.76	82.82±1.52
25	1.0	84.54±1.62	83.23±0.82	82.75±0.24	81.63 ±1.75	**86.63 ±0.73**	84.39 ±0.76	85.71±0.38	85.93 ±1.37	81.87±0.91	81.72±1.65
	Performance on Plateau problem										
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
25	0.1	86.47±0.43	85.37±0.97	84.72±0.94	86.73 ±0.58	**87.82 ±1.35**	82.71 ±0.49	78.64±0.25	74.69 ±0.75	82.83±1.02	81.72±0.39
25	0.2	85.51±0.73	84.42±1.24	83.72±0.58	72.28 ±0.54	**86.76 ±1.43**	82.71 ±1.34	78.92±0.67	82.57 ±1.27	80.49±1.64	79.76±0.61
25	0.5	84.71±0.71	83.83±1.41	83.65±1.59	82.91 ±0.82	**82.56 ±1.51**	72.63 ±0.93	71.52±0.63	75.83 ±1.47	74.71±1.53	80.29±0.78
25	1.0	85.54±0.62	84.56±0.67	83.52±0.93	81.25 ±0.83	**85.92 ±1.32**	83.64 ±0.59	79.38±0.75	75.58 ±1.27	83.65±0.38	85.52±1.43
	Performance on Deceptive problem										
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
25	0.1	87.68±1.93	82.48±0.97	85.67±0.82	86.72 ±0.65	**86.52 ±1.31**	85.83 ±0.42	81.43±0.57	83.94 ±1.28	82.35±0.27	80.72±0.94
25	0.2	86.65±0.68	83.35±0.91	85.62±0.96	84.78 ±1.08	**88.41 ±1.28**	83.71 ±0.85	69.83±0.23	75.73 ±0.68	86.57±0.48	83.32±1.13
25	0.5	85.84±0.72	80.54±1.09	83.75±1.02	84.45 ±1.26	**85.82 ±1.36**	75.16 ±0.65	82.49±0.75	84.36 ±1.10	81.68±0.73	82.65±0.26
25	1.0	88.58±0.83	81.82±0.13	83.76±1.02	82.85 ±1.15	**89.65 ±1.22**	82.64 ±0.73	85.17±0.58	67.72 ±1.65	63.56±0.58	72.25±0.67

Table 6.5: Comparison results on dynamic problems ($\tau = 50$)

		Performance on oneMax problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
50	0.1	89.56±0.01	87.35±0.15	80.18±0.23	88.52 ±0.24	**90.56±0.77**	88.28 ±0.22	70.12±0.19	73.18 ±0.14	68.38±0.17	62.45±0.12
50	0.2	87.62±0.23	74.35±0.47	86.65±0.15	81.73 ±0.25	**88.48±1.64**	87.15 ±0.32	82.73±0.52	87.37 ±0.28	85.62±0.43	76.82±0.53
50	0.5	85.76±0.08	84.37±0.04	83.18±0.12	80.53 ±0.16	**86.91±0.78**	85.62 ±0.42	84.87±0.53	82.25 ±0.16	80.44±0.29	85.11±0.48
50	1.0	82.32±0.45	81.31±0.38	80.18±0.45	82.78 ±0.65	**84.24±0.63**	82.37 ±0.76	81.53±0.64	80.34 ±0.44	78.32±0.65	73.67±0.72
		Performance on RoyalRoad problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
50	0.1	89.76±0.76	75.63±0.52	83.23±0.56	76.54 ±0.78	**91.92 ±1.64**	90.35 ±1.43	87.92±0.76	85.26 ±0.82	71.84±0.85	75.78±0.24
50	0.2	87.54±0.75	86.12±0.64	88.14±0.64	73.47 ±0.18	**88.83 ±1.13**	70.65 ±0.32	71.56±0.28	80.75 ±0.83	70.43±0.54	82.63±0.32
50	0.5	88.42±0.25	87.58±1.32	64.67±0.48	62.78 ±0.82	**90.51 ±0.62**	82.61 ±1.22	78.84±0.32	81.76 ±0.67	79.18±1.82	82.54±1.45
50	1.0	91.34±1.74	82.82±0.72	82.34±0.65	81.53 ±1.74	**91.78 ±1.72**	90.76 ±0.78	87.73±0.86	82.84 ±1.64	74.65±0.83	85.97±1.92
		Performance on Plateau problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
50	0.1	90.52±0.14	83.74±0.58	82.64±0.45	68.24 ±0.82	**91.68 ±1.82**	89.25 ±0.57	87.53±0.26	88.54 ±1.53	86.35±1.67	84.52±0.82
50	0.2	85.32±0.63	83.45±0.73	88.89±0.54	68.67 ±0.87	**90.27 ±1.56**	82.78 ±1.61	73.64±0.62	87.32 ±1.45	89.72±1.21	84.54±0.45
50	0.5	87.23±0.36	86.53±1.54	86.53±1.56	85.87 ±0.83	**89.64 ±1.75**	64.93 ±0.35	69.78±0.63	74.48 ±1.73	79.43±1.43	85.58±0.98
50	1.0	86.74±0.53	85.43±0.48	71.43±0.54	75.54 ±0.82	**88.76 ±1.34**	85.65 ±0.75	87.57±0.59	73.89 ±1.16	86.16±0.43	87.74±0.56
		Performance on Deceptive problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
50	0.1	92.81±1.76	85.95±0.93	85.63±0.76	84.73 ±0.78	**92.95 ±1.62**	89.75 ±0.42	87.67±0.75	88.32 ±1.58	87.72±0.54	91.65±1.54
50	0.2	91.52±0.56	90.54±0.87	90.87±0.87	90.67 ±1.56	**92.49 ±1.82**	79.65 ±0.42	81.45±0.23	87.49 ±0.58	75.68±0.73	91.27±1.42
50	0.5	86.93±0.47	84.76±1.73	83.86±1.95	82.56 ±1.58	**87.37 ±1.76**	86.73 ±0.67	85.75±0.93	84.78 ±1.79	82.78±0.84	83.87±0.35
50	1.0	90.68±1.68	87.82±1.94	88.85±1.82	82.76 ±1.28	**90.89 ±1.85**	88.32 ±0.94	87.75±0.45	73.76 ±1.75	87.38±0.48	89.35±1.73

Table 6.6: Comparison results on dynamic problems ($\tau = 100$)

		Performance on oneMax problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
100	0.1	91.57±1.32	91.35±1.17	89.26±1.16	87.60 ±0.32	**92.48±2.24**	90.23 ±0.27	88.78±0.32	82.23 ±0.24	89.53±0.42	88.25±0.37
100	0.2	90.74±0.17	89.43±0.45	87.47±0.15	88.75 ±0.64	**91.95±1.68**	86.65 ±0.38	83.82±0.37	90.56 ±0.46	85.74±0.38	84.76±0.43
100	0.5	88.76±0.34	87.35±0.21	85.45±0.23	86.45 ±0.38	**90.85±1.94**	84.52 ±0.49	87.92±0.53	83.17 ±0.22	89.28±0.27	87.65±0.72
100	1.0	90.62±0.45	87.63±0.56	89.37±0.54	88.52 ±0.48	**90.93±1.84**	87.69 ±0.83	85.87±0.83	89.25 ±0.29	84.37±0.56	89.94±1.83
		Performance on RoyalRoad problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
100	0.1	89.75±0.14	87.37±0.85	88.23±0.59	89.72 ±0.92	**90.64 ±1.85**	79.85 ±0.54	85.74±0.86	83.27 ±0.85	82.71±0.62	81.76±0.56
100	0.2	90.59±0.87	87.45±0.82	85.54±0.63	82.85 ±0.65	**91.72 ±1.34**	85.85 ±0.46	84.56±0.32	84.73 ±0.72	86.28±0.76	89.87±0.41
100	0.5	91.67±0.45	89.57±1.42	86.65±1.49	89.86 ±0.75	**92.03 ±1.93**	91.45 ±1.45	90.26±0.39	88.59 ±0.51	92.12±1.85	81.16±1.18
100	1.0	88.44±1.67	85.73±0.87	89.48±0.71	84.38 ±1.43	**90.28 ±1.61**	89.61 ±0.78	85.74±0.83	85.78 ±1.62	85.68±0.53	87.85±1.64
		Performance on Plateau problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
100	0.1	91.67±0.37	90.53±0.45	89.67±0.56	89.48 ±0.73	**91.94 ±1.65**	90.45 ±0.58	89.58±0.42	90.76 ±1.94	85.67±1.67	88.68±0.73
100	0.2	90.35±0.64	88.47±0.83	87.83±0.32	80.48 ±0.68	**91.81 ±1.94**	90.95 ±1.57	82.94±0.64	87.39 ±1.36	85.76±1.78	82.73±0.25
100	0.5	90.48±0.45	88.76±1.56	76.87±1.36	87.67 ±0.87	**90.73 ±1.78**	56.84 ±0.53	82.94±0.79	88.35 ±1.72	80.56±1.53	78.58±0.94
100	1.0	88.56±0.12	86.56±0.78	85.95±0.49	81.25 ±0.59	**89.72 ±2.12**	82.64 ±0.73	88.98±1.73	85.84 ±1.65	87.36±0.56	84.48±0.67
		Performance on Deceptive problem									
τ	ρ	AHMA	DDDGA	CHMA	RIGA	SSM	HSA-I	CPSO-MHS	HSA-M	mQSO	HSA-MI
100	0.1	89.93±1.76	86.43±0.75	88.61±0.53	76.72 ±0.83	**90.84 ±1.68**	83.48 ±0.39	89.67±0.74	88.52 ±1.54	81.82±1.87	79.89±0.75
100	0.2	87.72±0.87	86.52±0.87	86.94±0.73	88.71 ±1.68	**89.98 ±1.84**	82.84 ±0.45	83.45±0.63	88.94 ±0.78	83.71±0.63	82.57±0.87
100	0.5	88.89±0.87	84.78±1.86	86.58±1.69	85.45 ±1.57	**90.92 ±1.93**	89.75 ±0.92	85.57±0.95	85.84 ±1.74	89.63±0.84	89.76±0.34
100	1.0	90.78±0.86	88.83±1.94	89.73±1.65	79.73 ±1.53	**91.84 ±2.02**	90.16 ±0.74	87.37±0.76	82.88 ±1.58	82.27±0.51	80.54±0.94

6.4 Conclusion

This chapter presented results obtained from the experiments conducted on several problems by the proposed methods. All the proposed methods performed better when compared with the existing approaches. The proposed efficient choice function made HHs more intelligent in choosing good heuristics which, in turn, yielded good solutions when mapped onto problems. The memetic search was improved by the introduction of a spy search method, which performed a high quality search based on gathered information about the search environment and its elements. The proposed non-covariance function with the Gaussian process regression enabled dynamic HHs to solve dynamic and complex problems. The proposed method yielded good performance in dynamic HHs.

Chapter 7

Conclusion and Future Work

This work aimed to improve the search and selection performance of hyper-heuristics in optimization. As a result, the following three contributions were presented in this thesis.

1. An improved heuristic selection mechanism called the efficient choice function was proposed. The mechanism addressed the drawbacks of the existing choice function, as explained in Chapter 1. The ability of a choice function to search for an optimal heuristic was enhanced, as shown in the improved performance of an efficient choice function.

2. The hyper-heuristics studied in this work was based on a memetic search. Therefore, further improvements were made on a memetic algorithm. A spy search method was proposed which enhanced the ability of the algorithm to yield good outcomes, as presented in the results. The proposed method was tested on several dynamic problems and comparisons with the existing methods showed better performance by the proposed method.

3. An investigation was conducted on dynamic hyper-heuristics where there was moving optima. A non-covariance function with the Gaussian process regression was proposed. This is a predictive measure based on patterns of the search space, which helped dynamic hyper-heuristics to solve dynamic and complex problems. Results showed a better performance by the proposed hyper-heuristics than by the existing hyper-heuristics.

The future extensions of this work can include, investigating multiple-point search methods so as to combine one-point and multiple-point methods for better performance of hyper-heuristics. Creating more functions for dynamic hyper-heuristics to track moving optima in their abstraction search spaces is also recommended.

Bibliography

[1] E. S. A. Dowsland and E. K. Burke, "A simulated annealing hyper-heuristic for determining shipper sizes for storage and transportation," *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.

[2] P.Cowling, G.Kendall, and E.Soubeiga, "A hyperheuristic approach for scheduling a sales summit," in *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, (Konstanz, Germany), pp. 176–190, 2000.

[3] J.H.Drake, E.Ozcan, and E.K.Burke, "An improved choice function heuristic selection for cross domain heuristic search," in *Parallel Problem Solving from Nature-PPSN XII* (C.C.Coello, V.Cutello, K.Deb, S.Forrest, G.Nicosia, and M.Pavone, eds.), pp. 307–316, 2012.

[4] C.Cotta, M.Sevaux, and K.Srensen, *Adaptive and Multilevel Metaheuristics: Studies in Computational Intelligence*. Springer, 2008.

[5] G.Kendall, E.Soubeiga, and P.Cowling, "Choice function and random hyperheuristics," in *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pp. 667–671, 2002.

[6] M.Kampouridis, "An initial investigation of choice function hyper-heuristics for the problem of financial forecasting," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2406–2413, 2013.

[7] B.Kiraz, A.S.Uyar, and E.Ozcan, "Selection hyper-heuristics in dynamic environments," in *European Conference on the Applications of Evolutionary Computation* (C.DiChio, ed.), pp. 314–323, 2011.

[8] P.Cowling, G.Kendall, and E.Soubeiga, "Hyperheuristics: A tool for rapid prototyping in scheduling and optimization," in *Proceeding of Applications of Evolutionary Computing:1718 Journal of the Operational Research Society, No. 12 Workshops.* (E. M. S.Cagoni, J.Gottlieb and R.Goenther, eds.), (Kinsale, Ireland), pp. 1–10, 2002.

[9] P.Cowling, G.Kendall, and E.Soubeiga, "A hyperheuristic approach to scheduling a sales summit.," in *In Proceedings of the Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling. PATAT 2000, Lecture Notes in Computer Science* (E.K.Burke and W.Erben, eds.), (Konstanz,Germany), pp. 176–190, 2000.

[10] P.Rattadilok, "An investigation and extension of a hyper-heuristic framework," *Informatica*, vol. 34, pp. 523–534, 2010.

[11] R. Goncalves, J.N.Kuk, C.P.Almeida, and S.M.Venske, "Moea/d-hh:a hyper-heuristic for multi-objective problems," in *Evolutionary Multi-Criterion for Optimization* (A.Gaspar-Cunha, C.H.Antunes, and C.C.Coello, eds.), pp. 97–115, 2015.

[12] E.K.Burke, G. E.Hart, J.Newall, P.Ross, and S.Schulenburg, "Hyperheuristics: An emerging direction in modern search technology," in *Handbook of Metaheuristics* (F.Glover and G.Kochenberger, eds.), (Kluwer: Dordrecht, MA), pp. 267–271, 2003.

[13] E.Hart and K.Sim, "A hyper-heuristic ensemble method for static job-shop scheduling," *Evolutionary Computation*, vol. 24, pp. 609–635, 2016.

[14] E.K.Burke, M.Gendreau, M.Hyde, G.Kendall, G.Ochoa, E.Ozcan, and R.Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 2013.

[15] G.Ochoa, M.Hyde, J.-R. T.Curtois, J.Walker, M.Gendreau, A. G.Kendall, S.Petrovic, and E.K.Burke, "Hyflex: A benchmark framework for cross-domain heuristic search," in *Proceedings of the 12th European Conference on Evolutionary Computation in Combinatorial Optimization*, (Berlin), pp. 136–147, 2012.

[16] J.G.Marin-Blazquez and S.Schulenburg, "A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients," in *Proceedings of the 8th International Workshop on Learning Classifier Systems (IWLCS05)* (T.Kovacs, X.Llora, K.Takadama, P.L.Lanzi, W.Stolzmann, and S.W.Wilson, eds.), pp. 193–218, 2005.

[17] P.Ross, J.G.Marin-Blazquez, S.Schulenburg, and E.Hart, "Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'2003)* (E.Cantu-Paz, J.A.Foster, K.Deb, L.D.Davis, R.Roy, U.O'Reilly, H.Beyer, R.Standish, G.Kendall, S.Wilson, M.Harman, J.Wegener, D.Dasgupta, M.A.Potter, A.C.Schultz, K.A.Dowsland, N.Jonoska, and J.Miller, eds.), pp. 1295–1306, 2003.

[18] K.Edmund, T.Burke, M.Curtois, G.Hyde, G. Kendall, G.Ochoa, S.Petrovic, and J.A.VazquezRodriguez, "Hyflex: A flexible framework for the design and analysis of hyper-

heuristics," in *Multidisciplinary International Scheduling Conference (MISTA 2009)*, pp. 790–797, 2009.

[19] H.Fisher and G.L.Thompson, *Probabilistic learning combinations of local job-shop scheduling rules*, ch. Industrial Scheduling, pp. 225–251. Springer, 1963.

[20] E.K.Burke, M.Hyde, G.Kendall, G.Ochoa, E.Ozcan, and J.Woodard, "A classification of hyper-heuristic," in *Handbook of Metaheuristics, International Series in Operations Research and Management Science*, pp. 449–468, 2010.

[21] E.K.Burke, B.McCollum, A.Meisels, S.Petrovic, and R.Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177–192, 2007.

[22] M.Maashi, E.Ozcan, and G.kendall, "A multi-objective hyper-heuristic based on choice function," *Expert Systems with Applications*, vol. 41, pp. 4475–4493, 2014.

[23] A.Kheiri, M.Musir, and E.Ozcan, "Ensemble move acceptance in selection hyper-heuristics," in *International Symposium on Computer and Information Systems* (T.Czachorski, ed.), pp. 21–29, 2016.

[24] E.Ozcan, B.Bilgin, and E.E.Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent data analysis*, vol. 12, no. 1, pp. 3–23, 2008.

[25] B.Bilgin, E.Ozcan, and E.E.Korkmaz, "An experimental study on hyper-heuristics and exam timetabling," in *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)* (E.K.Burke and H.Rudov, eds.), (Brno,Czech Republic), pp. 394–412, 2007.

[26] R.Dawkins, *The Selfish Gene*. Clarendon Press, 1976.

[27] M. G. Norman and P. Moscato, "A competitive cooperative complex combinatorial search," in *Proceedings of the Joint Conference on Informatics and Operations Research*, pp. 15–29, 1991.

[28] G.J.Park, *Design of experiments*, ch. Analytic Methods for Design Practice, pp. 309–391. Springer London, 2007.

[29] Y.Jin and J.Branke, "Evolutionary optimization in uncertain environments-a survey," *Journal of the IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 303–317, 2005.

[30] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Dordrecht, MA: Kluwer, 2002.

[31] N.Sabar, M.Ayob, R.Qu, and G.Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, 2012.

[32] M.Ayob, A.M.A.Malik, S.Abdullah, A.R.Hamdan, G.Kendall, and R.Qu, "Solving a practical examination timetabling problem: A case study," in *Proceedings of the International Conference on Computational Science and Applications* (O.Gervasi and M.Gavrilova, eds.), vol. 4707, pp. 611–624, 2007.

[33] J.Mlejnek and J.Kubalik, "Evolutionary hyper-heuristic for capacitated routing problem," in *15th annual conference companion on Genetic and evolutionary computation* (C.Blum, ed.), pp. 219–220, 2013.

[34] Z.A.Aziz and G.Kendall, "An investigation of an ant-based hyper-heuristic for the capacitated vehicle routing problem," in *Multidisciplinary International Conference on Scheduling: Theory and Applications* (E.Burke and H.Rudova, eds.), pp. 823–826, 2009.

[35] P.Ross and J.G.Marin-Blazquez, "Constructive hyper-heuristics in class timetabling," in *IEEE Congress on Evolutionary Computation*, pp. 1493–1500, 2005.

[36] K.Socha, J.Knowles, and M.Sampels, "A max-min ant system for the university course timetabling problem," in *Third International Workshop, ANTS* (M.Dorigo and G.D.Caro, eds.), pp. 1–13, 2002.

[37] B.Paechter, R.C.Rankin, A.Cumming, and T.C.Fogarty, "Timetabling the classes of an entire university with an evolutionary algorithm.," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature* (M. A.E.Eiben, T.Back and H.Schwefel, eds.), pp. 865–874, 1998.

[38] N.Pillay and W.Banzhaf, "A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem," *European Journal of Operational Research*, vol. 197, pp. 1–10, 2008.

[39] M.W.Carter, G.Laporte, and S.Y.Lee, "Examination timetabling algorithmic strategies and applications," *The Journal of Operational Research Society*, vol. 47, no. 3, pp. 373–383, 1996.

[40] E.K.Burke, J.Li, and R.Qu, "Data mining: An aid towards more efficient hyper-heuristic search," in *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling* (E.Burke and H.Rudova, eds.), pp. 1–4, 2008.

[41] J.H.Drake, M.Hyde, K.Ibrahim, and E.Ozcan, "A genetic programming hyper-heuristic for the multidimensional knapsack problem," *Kybernetes*, vol. 43, pp. 1500–1511, 2014.

[42] F.Thabtah and P.Cowling, "Mining the data from a hyper-heuristic approach using associative classification," *Expert Systems with Applications*, vol. 34, no. 2, pp. 1093–1101, 2008.

[43] K.Chakhlevitch and P.Cowling, "Choosing the fittest subset of low level heuristics in a hyperheuristic framework," in *Proceedings of 5th European Conference on Evolutionary Computation in Combinatorial Optimization* (G.R.Raidl and J.Gottlieb, eds.), (New York), pp. 25–33, 2005.

[44] D.Corne and P.Ross, "Peckish initialisation strategies for evolutionary timetabling," in *Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science* (E.Burke and P.Ross, eds.), vol. 1153, pp. 227–240, 1995.

[45] S.Abdullah, N.R.Sabar, M.Zakree, and B.McCollum, "A constructive hyper-heuristics for rough set attribute reduction," in *10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 1032–1035, 2010.

[46] J.Rankhambe and S.Kavita, "Optimization of examination timetable using harmony search hyper-heuristics (hsh)," *International Journal of Computer Science and Information Technologies*, vol. 5, pp. 6719–6723, 2014.

[47] V. Cerny, "A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm.," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.

[48] P.Garrido and C.Castro, "Stable solving of cvrps using hyper-heuristics," *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, vol. 6622, pp. 255–262, 2009.

[49] R.Raghavjee and N.Pillay, "A genetic algorithm selection perturbative hyper-heuristic for solving the school timetabling problem," *Orion*, vol. 31, pp. 39–60, 2015.

[50] I.Maden, S.Uyar, E.Ozcan, and N.B.Nottingham, "Landscape analysis of simple perturbative hyper-heuristics," in *Proceeding of the 15th International Conference on Soft computing* (A. N.S.Herman, S.M.Shamsuddin, ed.), pp. 1–7, 2009.

[51] C.Rae and N.Pillay, "Investigation into an evolutionary algorithm hyper-heuristic for the nurse rostering problem," in *10th International Conference of the Practice and Theory of Automated Timetabling* (E. E.Ozcan and B.McCollum, eds.), pp. 26–29, 2014.

[52] E.K.Burke, M.Gendreau, M.Hyde, K.Kendall, G.Ochoa, E.Ozcan, and R.Qu, "Hyper-heuristics:a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[53] R.Bai, E.Burke, G.Kendall, J.Li, and B.McCollum, "A hybrid evolutionary approach to the nurse rostering problem.," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 580–590, 2010.

[54] E.K.Burke, G.Kendall, and E.Soubeiga, "A tabu search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[55] E.Ozcan, M.Misir, and E. K.Burke, "A self-organising hyper-heuristic framework," in *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications* (J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, eds.), pp. 784–787, 2009.

[56] 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* (C.Mai, ed.), pp. 132–141, 2003.

[57] G.Kendall and M.Mohamad, "Channel assignment optimization using a hyperheuristic," in *Cybernetics and intelligent systems*, (Singapore), pp. 791–796, 2004.

[58] G.Laporte and I.H.Osman, "Routing problems:a bibliography," *Annals Operations Research*, vol. 61, pp. 227–262, 1995.

[59] N.Pillay, "Evolving heuristics for the school timetabling problem," in *Proceedings of the 2011 IEEE Conference on Intelligent Computing and Intelligent Systems* (F.Wang and S.Berte, eds.), vol. 3, pp. 281–286, 2011.

[60] E.K.Burke, M.Hyde, G.Kendall, and J.Woodward, "Automatic heuristic generation with genetic programming: Evolving jack-of-all-trades or a master of one," in *Proceedings of the 9th ACM Genetic and Evolutionary Conference* (H.Lipson, ed.), pp. 1559–1565, 2007.

[61] R.Kumar, A.H.Joshi, K.K.Banka, and P.I.Rockett, "Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming," in *Proceedings from Genetic and Evolutionary Computation Conference* (C.Yan and M.Keijzer, eds.), pp. 1227–1234, 2008.

[62] N.Pillay, "A study of evolutionary algorithm selection hyper-heuristics for the one-dimensional packing problem," *South African Computer Journal*, vol. 48, pp. 31–40, 2012.

[63] J.A.Vazquez-Rodriquez, G.Ochoa, T.Curtols, and Hyde, "A hyflex module for the permutation flow shop problem," tech. rep., University of Nottingham, Nottinghamshire, 2010.

[64] P.C.Chen, G.Kendall, and G.V.Berghe, "An ant based hyper-heuristic for the travelling tournament problem.," in *Proceedings of IEEE Symposium of Computational Intelligence in Scheduling*, (Hawaii), pp. 19–26, 2007.

[65] L.Han and G.Kendall, *Guided Operators for a Hyper-Heuristic Genetic Algorithm*, ch. AI 2003: Advances in Artificial Intelligence Volume 2903 of the series Lecture Notes in Computer Science, pp. 807–820. Springer Berlin Heidelberg, 2003.

[66] S. W. Ju, C. Y. Han, H. M. Lin, and C. Tsung-Che, "Ntnu-ma: A memetic algorithm-based hyperheuristic for the chesc 2014 competition," in *CHeSC 2014, OR53 Annual Conference*, pp. 2–7, 2014.

[67] Y. Lei, M. Gong, L. Jiao, and Y.Zuo, "A memetic algorithm based on hyper-heuristics for examination timetabling problems," *International Journal of Intelligent Computing and Cybernetics*, vol. 8, pp. 139–151, 2015.

[68] E. Segredo, C. Segura, and C. Leon, "Memetic algorithms and hyperheuristics applied to a multiobjectivised two-dimensional packing problem," *Journal of Global Optimization*, vol. 58, pp. 769–794, 2014.

[69] C. Leon, G. Miranda, and C. Segura, "A memetic algorithm and a parallel hyperheuristic island-based model for a 2d packing problem," in *Proceeding GECCO '09 Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1371–1378, 2009.

[70] J.E.Smith, "Coevolving memetic algorithms: A review and progress report," *IEEE Transactions on Systems Man and Cybernetics*, vol. 37, pp. 6–17, 2007.

[71] M. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memetic Computing*, vol. 1, pp. 205–219, 2009.

[72] A. Elyasaf, *Evolving Hyper-Heuristics using Genetic Programming*. PhD thesis, University of the Negev, 2014.

[73] N. Krasnogor, "Self generating metaheuristics in bioinformatics: The proteins structure comparison case," *Genetic Programming and Evolvable Machines*, vol. 5, pp. 181–201, 2004.

[74] P. Merz and B. Freisleben, "A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2063–2070, 1999.

[75] P. Merz and B. Freisleben, "Memetic algorithms for the traveling salesman problem," *Computer Systems*, vol. 13, pp. 297–345, 2001.

[76] M.Bader-El-Den and R.Poli, "An incremental approach for improving local search heuristics," in *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science*, pp. 194–219, 2008.

[77] M.Yannakakis, "The analysis of local search problems and their heuristics," in *Proceedings of the 7th Annual Symposium of Theoretical Aspects of Computer Science* (C.Choffrut and T.Lengauer, eds.), pp. 298–311, 1990.

[78] D.H.Wolpert, "The lack of a prior distinctions between learning algorithms and the existence of a priori distinctions between learning algorithms," *Neural Computation*, vol. 8, pp. 1341–1421, 1996.

[79] C.Schaffer, "A conservation law for generalization performance," in *Proceedings of the 11th International Conference on Machine Learning*, pp. 259–265, 1994.

[80] D.Wolpert and W.Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, 1997.

[81] P.Cowling, G.Kendall, and E.Soubeiga, "A parameter-free hyperheuristic for scheduling a sales summit," in *In Proceedings of the 4th Metaheuristic International Conference.* (J.P.Sousa and M.G.C.Resende, eds.), (Porto, Portugal), pp. 127–131, 2001.

[82] E.K.Burke, M.Hyde, G.Kendall, G.Ochoa, E.Ozcan, and J.Woodward, *Handbook of metaheuristics, international series in operations research management science*, ch. A Classification of Hyper-heuristic Approaches, pp. 449–468. Springer, 2010.

[83] N.Fouladgar and S.Lotfi, "A brief review of solving dynamic optimization problems," *International Academic Journal of Science and Engineering*, vol. 2, pp. 26–33, 2015.

[84] D.B.Kell, "Scientific discovery as a combinatorial optimizational problem: How best to navigate the landscape of possible experiments," *Bioessays*, vol. 34, no. 3, pp. 236–244, 2012.

[85] M.B.Khan, D.Zhang, M.Jun, and Z.J.Li, "An intelligent search technique to train scheduling problem based on genetic algorithm," in *International Conference on Emerging Technologies*, pp. 593–598, 2006.

[86] G.Ming and H.Li, "An improved algorithm based on max-min for cloud task scheduling," in *Recent Advances in Computer Science and Information Engineering* (Z.Qian, L.Cao, W.Su, T.Wang, and H.Yang, eds.), pp. 217–223, 2012.

[87] E.U.Munir, J.Li, and S.Shi, "Qos sufferage heuristic for independent task scheduling in grid," *Information Technology Journal*, vol. 6, pp. 1166–1170, 2007.

[88] C. Tsai, W. Huang, and M. Chiang, "A hyperheuristic scheduling algorithm for cloud," *IEEE Transactions on Cloud Computing*, vol. 2, pp. 236–250, 2014.

[89] H.G.Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time dependent nonstationary environment," tech. rep., Technical Report AIC-90- 001, Naval Research Laboratory, Washington, USA, 1990.

[90] J.J.Grefenstette, "Genetic algorithms for changing environments," in *Proceedings of the Parallel Problem Solving From Nature II*, pp. 137–144, 1992.

[91] H.G.Cobb and J.J.Grefenstette, "Genetic algorithms for tracking changing environments," in *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 523–530, 1993.

[92] F.Vavak and T.C.Fogatry, "A comparative study of steady state and generational genetic algorithms for use in nonstationary environments," in *AISB Workshop on Evolutionary Computing*, pp. 297–304, 1996.

[93] S.H.Jacob and E.Yucesan, "Analyzing the performance of generalized hill climbing algorithms," *Journal of Heuristics*, vol. 10, pp. 387–405, 2004.

[94] J.Digalakis and K.Margaritis, "Performance comparison of memetic algorithms," *Applied Mathematics and Computation*, vol. 158, pp. 237–252, 2004.

[95] A.Prugel-Bennett, "When a genetic algorithm outperforms hill-climbing," *Theoretical Computer Science*, vol. 320, pp. 135–153, 2004.

[96] D.P.Leyden, "Modified quadratic hill-climbing with sas/iml," *Computer Science in Economics and Management*, vol. 4, pp. 15–31, 1991.

[97] B.Sareni and L.Krahenbuhl, "Fitness sharing and niching methods," *IEEE Transactions on Evolutionary Computation*, vol. 2, pp. 97–108, 1998.

[98] J.H.Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan, 1975.

[99] D.E.Goldberg and J.Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proceedings of the 2nd International Conference of Genetic Algorithms*, pp. 41–49, 1987.

[100] K.A.DeJong, *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, USA, 1975.

[101] N.Casas, "Genetic algorithm for multimodal optimization:a review," *CoRR Journal*, vol. 2, pp. 1–7, 2015.

[102] O.J.Mengshoel and D.E.Goldberg, "Probabilistic crowding:deterministic crowding with probabilistic replacement," in *Proceedings of the Genetic and Evolutionary Computation Conference* (W.Banzhaf, J.Daida, A.E.Eiben, M.H.Garzon, V.Honavar, M.Jakiela, and R.E.Smith, eds.), pp. 409–416, 1999.

[103] H.Wang, D.Wang, and S.Yang, "A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems," *Soft Computing*, vol. 13, pp. 763–780, 2009.

[104] G.Lin, X.Lu, and L.Kang, "Search direction made evolution strategies faster," in *Proceedings of the 4th International Computational Intelligence and Intelligent Systems*, pp. 146–155, 2009.

[105] R.W.Morrison and K.A.DeJong, "A test problem generator for non-stationary environments," in *Congress on Evolutionary Computation*, vol. 3, pp. 2047–2053, 1999.

[106] S.Yang, "Non-stationary problem optimization using the primal-dual genetic algorithm," in *Proc. of the 2003 Congr. on Evol. Comput.*, vol. 3, pp. 2246–2253, 2003.

[107] S.Yang and X.Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Computing*, vol. 9, no. 11, pp. 815–834, 2005.

[108] S. F. M. Mitchell and J. H. Holland, "The royal road for genetic algorithms: fitness landscapes and ga performance," in *Proc. of the 1st European Conf. on Artificial Life*, pp. 245–254, 1992.

[109] S.Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proc. of the 2005 Genetic and Evol. Comput. Conference*, vol. 2, pp. 1115–1122, 2005.

[110] S.Forrest and M.Mitchell, "Relative building-block fitness and the building-block hyperthesis," in *Proceedings of the Foundations of Genetic Algorithms II* (D. Whitley, ed.), pp. 109–126, 1992.

[111] H. Wang, D. Wang, and S. Yang, "A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems," *Soft Computing*, vol. 13, pp. 763–780, 2009.

[112] S.Yang, "On the design of diploid genetic algorithms for problem optimization in dynamic environments," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pp. 1362–1369, 2006.

[113] A.M.Turky, S.Abdullah, and N.R.Sabar, "Meta-heuristic algorithm for binary dynamic optimisation problems and its relevancy to timetabling," in *Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling*, pp. 26–29, 2014.

[114] G.Zhang and Y.Li, "A memetic algorithm for global optimization of multimodal nonseparable problems," in *IEEE Transactions on Cybernetics*, vol. 46, pp. 1375–1387, 2016.

[115] A.M.Turky and S.Abdullah, "A multi-population harmony search algorithm with external archive for dynamic optimization problems," *Information Systems*, vol. 172, pp. 84–95, 2014.

[116] J.Denzinger, M.Fuchs, and M.Fuchs, "High performance atp systems by combining several ai methods," in *Proceeding IJCAI'97 Proceedings of the 15th international joint conference on Artificial intelligence*, pp. 102–107, 1997.

[117] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Proceedings of the Third International Conference, Practice and Theory of Automated Timetabling III* (E.Burke and E.Erben, eds.), pp. 176–190, Springer-Berlin Heidelberg, 2001.

[118] N.Wiener, *Extrapolation, Interpolation and Smoothing of Stationary Time Series.* MIT Press, 1949.

[119] G.Matheron, "The intrinsic random functions and their applications," *Advances in Applied Probability*, vol. 5, pp. 439–468, 1973.

[120] M.Ebden, "Gaussian processes for regression." `http://www.robots.ox.ac.uk/~mebden/reports/GPtutorial.pdf`. Accessed:2015-09-17.

[121] C.J.Paciorek and M.J.Schervish, "Spatial modelling using a new class of nonstationary covariance functions," *Environmetrics*, vol. 17, no. 5, pp. 483–506, 2006.

[122] K.Weicker, *Evolutionary Algorithms and Dynamic Optimization Problems.* PhD thesis, PhD thesis, University of Stuttgart, Institut fur Formale Methoden der Informatik der, Germany, 2003.

[123] S. Gharan and A. Saberi, "The asymmetric traveling salesman problem on graphs with bounded genus," in *SODA11 Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pp. 967–975, 2011.

[124] P.Collard, C.Escazut, and A.Gaspar, "The asymmetric traveling salesman problem on graphs with bounded genus," in *Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence*, pp. 2–9, 1996.

[125] H.N.Psaraftis, "Dynamic vehicle routing: Status and prospects," *Annals of Operations Research*, vol. 61, pp. 143–164, 1995.

[126] C.Li, M.Yang, and L.Kang, "A new approach to solving dynamic traveling salesman problems," in *Asia-Pacific Conference on Simulated Evolution and Learning* (T.Wang, X.Li, S.Chen, X.Wang, H.Abbass, H.Iba, G.Chen, and X.Yao, eds.), pp. 236–243, 2006.

[127] S.Ray, S.Pal, and S.Bandyopadhyay, "Genetic operators for combinatorial optimization in tsp and microarray gene ordering," *Applied Intelligence*, vol. 26, pp. 183–195, 2007.

[128] J.K.Lenstra and A.H.R.Kan, "Some simple applications of the traveling salesman problem," *Operational Research Quarterly*, vol. 26, no. 4, pp. 717–733, 1975.

[129] D.Applegate, R.Bixby, V.Chvatal, and W.Cook, "On the solution of traveling salesman problems," *Mathematica*, vol. 3, pp. 645–656, 1998.

[130] G.B.Dantzig and J.R.Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[131] G.Reinelt, *Tsplib, a library of sample instances for the tsp*, 2008 (accessed August 10, 2015).

[132] G.Clarke and J.W.Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, no. 4, pp. 568–582, 1964.

[133] SINTEF, *Vrptw benchmark problems, on the sintef transport optimisation portal*, 2012 (accessed August 4, 2015).

[134] G.Laporte, M.Gendreau, J.Potvin, and F.Semet, "Classical and modern heuristics for the vehicle routing problem.," in *International Transactions in Operational Research*, pp. 285–300, 2000.

[135] K. Dowsland, "Nurse scheduling with tabu search and strategic oscillation," *European Journal of Operational Research*, vol. 106, pp. 393–407, 1998.

[136] T.Curtois, *Staff rostering benchmark data sets*, 2009 (accessed August 10, 2015).

[137] A.Ikegami and A.Niwa, "Subproblem-centric model and approach to the nurse scheduling problem," *Mathematical Programming*, vol. 97, no. 3, pp. 517–541, 2003.

[138] P.Cowling, G.Kendall, and E.Soubeiga, "Hyperheuristics: A robust optimisation method applied to nurse scheduling," in *PPSN 2002 LNCS* (J.J.M.Guervos, P.A.Adamidis, H.G.Beyer, J.L.Fernandez-Villacanas, and H.P.Schwefel, eds.), pp. 851–860, 2002.

[139] E.Taillard, "Benchmarks for the basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[140] R.Ruiz and T. Stutzle, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives.," *European Journal of Operational Research*, vol. 187, no. 3, pp. 1143–1159, 2008.

[141] S.Martello and P.Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley Sons, Inc., 1990.

[142] M.Hyde, G. Ochoa, T. Curtois, and J. Vazquez-RodrÃguez, "A hyflex module for the one dimensional bin-packing problem," tech. rep., Technical Report,School of Computer Science, University of Nottingham, 2009.

[143] E.K.Burke, M.Hyde, and G.Kendall, "Evolving bin packing heuristics with genetic programming," in *Proceedings of Parallel Problem Solving from Nature (PPSN 2006)* (T.P.Runarsson, H.Beyer, E.K.Burke, J.J.M.Guervos, L.D.Whitley, and X.Yao, eds.), pp. 860–869, 2006.

[144] J. Cordeau and G. Laporte, "The dial-a-ride problem (darp): variants, modeling issues and algorithms," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 89–101, 2003.

[145] Z.Zhang, *Hong Kong hospital data sets*, 2009 (accessed January 12, 2016).

[146] R.M.Jorgensen, J.Larsen, and K. Bergvinsdottir, "Solving the dial-a-ride problem using genetic algorithms," *Journal of the Operational Research Society*, vol. 58, no. 10, pp. 1321–1331, 2007.

[147] Y.Kaya, M.Uyar, and R.Tekin, "A novel crossover operator for genetic ring crossover," *Global Journal of Technology*, vol. 1, pp. 1–4, 2014.

[148] G.Ochoa, M.Hyde, J.-R. T.Curtois, J.D.Walker, M.Gendreau, G.Kendall, B.McCollum, A. Parkes, S.Petrovic, and R.Qu, "Hyflex: A benchmark framework for cross-domain heuristic search," in *Proceedings of Evolutionary Computation in Combinatorial Optimization (Evo-COP 2012)* (J.Hao and M.Middendorf, eds.), pp. 136–147, 2012.

[149] W.V.Onsem, B.Demoen, and P.D.Causmaecker, "Learning a hidden markov model-based hyper-heuristic," in *Proceedings of the 9th International Conference on Learning and Intelligent Optimization* (L. C.Dhaenens and M.Marmion, eds.), pp. 74–88, 2015.

[150] H.Lourenco, O.C.Martin, and T.Stutzle, *Handbook of Metaheuristics*, ch. Iterated Local Search, pp. 320–353. Springer,US, 2003.

[151] G.Schrimpf, J.Schneider, H.Stamm-Wilbrandt, and G.Dueck, "Record breaking and optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, pp. 139–171, 2000.

[152] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculation by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 139–171, 2000.

[153] E.Soubeiga, *Development and application of hyperheuristics to personnel scheduling*. PhD thesis, PhD Thesis, Department of Computer Science, University of Nottingham, UK, 2003.

[154] E.Taillard, *Flow shop benchmark data sets*, 2010 (accessed August 11, 2015).

[155] M.Nawaz, E.Enscore, and I.Ham, "A heuristic algorithm for the m-machine, n -job flow-shop sequencing problem," *OMEGA-International Journal of Management Science*, vol. 11, no. 1, pp. 91–95, 1983.

[156] M.Hyde and G.Ochoa, "A hyflex competition instance summary," tech. rep., Technical Report, School of Computer Science, University of Nottingham, 2011.

[157] C.E.Rasmussen and H.Nickisch, "Gaussian processes for machine learning (gpml) toolbox," *The Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, 2010.

[158] G.Reinelt, "The tsplib symmetric traveling salesman problem instances." `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`. Accessed:2015-08-30.

[159] X.Geng, Z.Chen, W.Yang, D.Shi, and K.Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," *Applied Soft Computing*, vol. 11, pp. 3680–3689, 2011.