University of Nottingham

Move Acceptance in Local Search Metaheuristics for Cross-domain Heuristic Search

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

May 2021

Author:

Warren G. Jackson

Supervisors:

Dr Ender Özcan

Prof Robert I. John

Abstract

Many real-world combinatorial optimisation problems (COPs) are computationally hard problems and search methods are frequently preferred as solution techniques. Traditionally, an expert with domain knowledge designs, and tailors the search method for solving a particular COP. This process is usually expensive, requiring a lot of effort and time and often results in problem specific algorithms that can not be applied to another COP. Then, the domain expert either needs to design a new search method, or reconfigure an existing search method to solve that COP. This prompted interest into developing more general, problem-domain-independent high-level search methods that can be re-used, capable of solving not just a single problem but multiple COPs. The cross-domain search problem is a relatively new concept and represents a high-level issue that involves designing a single solution method for solving a multitude of COPs preferably with the least or no expert intervention. Cross-domain search methods are algorithms designed to tackle the cross-domain search problem. Such methods are of interest to researchers and practitioners worldwide as they offer a single off-the-shelf go-to approach to problem solving. Furthermore, if a cross-domain search method has a good performance, then it can be expected to solve 'any' given COP well and in a reasonable time frame. When a practitioner is tasked with solving a new or unknown COP, they are tasked with a decision-making dilemma. This entails the decision of what algorithm they should use, what parameters should be used for that algorithm, and whether any other algorithm can outperform it. A well designed cross-domain search method that performs well and does not require re-tuning can fulfil this dilemma allowing practitioners to find good-enough solutions to such problems. Researchers on the other hand strive to find high-quality solutions to these problems; however, such a cross-domain search method provides them with a good benchmark to which they can compare their solution methods to, and should ultimately aim to outperform.

In this work, move acceptance methods, which are a component of traditional search methods, such as metaheuristics and hyper-heuristics, are explored under a cross-domain search framework. A survey of the existing move acceptance methods as a part of local search metaheuristics is conducted based on the hyper-heuristic literature as solution methods to the cross-domain search problem. Furthermore, a taxonomy is provided for classifying them based on their design characteristics. The cross-domain performance of existing move acceptance methods, covering the taxonomy, is compared across a total of 45 problem instances spanning 9 problem domains, and the effects of parameter tuning versus choice of the move acceptance method are explored. A novel move acceptance method (HAMSTA) is proposed to overcome the shortcomings of the existing methods to improve the cross-domain performance of a local search metaheuristic. HAMSTA is capable of outperforming the cross-domain performances of existing methods that are re-tuned for each domain, despite itself using only a single cross-domain parameter configuration derived from tuning experiments that considers 2 instances each from 4 domains; hence, HAMSTA requires no expert intervention to re-configure it to perform well for solving multiple COPs with 37 problem instances unseen by HAMSTA, 25 of which are from unseen domains. HAMSTA is therefore shown to have the potential to fulfil the aforementioned decision-making dilemma.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I would like to take this opportunity to thank my supervisors Dr. Ender Özcan and Professor Robert I. John for their continued time, effort, and patience throughout my doctoral research. I would also like to give Dr. Ender Özcan a special mention as he inspired me to become involved in the field of optimisation during my undergraduate studies while at the University of Nottingham.

I express great appreciation to my examiners Dr Jonathan Thompson and Professor Dario Landa-Silva for their feedback and criticism of my research presented in the thesis for they have undoubtedly allowed me to strengthen my research for which I am grateful.

Recognition goes out to fellow colleagues within the Computational Optimisation and Learning (COL) lab (previously the ASAP research group) for their support and encouragement, and to the School of Computer Science itself.

I would also like to express gratitude to my parents Glenn and Sheila Jackson and to my significant other Chidchanok Sakdapanichkul for their support and encouragement throughout my doctorate.

Chapter 1

Introduction

The search methodologies used for tackling real-world combinatorial optimisation problems (COPs) [2], such as Examination Timetabling [3], High School Timetabling [4], and Vehicle Routing [5] are of interest to researchers and practitioners. *Metaheuristics* imposing 'a set of guidelines or strategies' based on a heuristic search framework [6] can be preferred over exact methods; while *exact methods* can find optimal solutions, their computational efforts (memory usage, flops, etc) can sometimes be orders of magnitude higher than *heuristic methods*. This is mainly due to the fact that many real-world problems are computationally hard to solve [7, 8] and exact methods can fail to produce acceptable solutions in a reasonable time frame. For example, a review of heuristic and exact algorithms [9] for solving the Quadratic Assignment Problem (QAP) emphasises this, stating that QAP instances most often discussed in the literature cannot be solved to optimality by exact methods when the problem sizes approach 30-40, whereas the heuristic search techniques can rapidly find high quality solutions. More recent work has gone into providing relaxed models for QAP stating that problem sizes over 30 are still not solvable by exact methods in any reasonable CPU time [10, 11].

Metaheuristics are classified in [6] into three categories; *local search* (e.g. Guided Local Search, Iterated Local Search, Late Acceptance, Record-to-record Travel, Simulated Annealing, and Tabu Search), *constructive* (e.g. GRASP and Ant-colony Optimisation), and *population-based* (e.g. Evolutionary Algorithms, Genetic Programming and Memetic Algorithms) [12]. Metaheuristics were categorised in [13] based on several distinctions; trajectory methods versus discontinuous methods, population-based versus single-point search, memory usage versus memoryless methods, one versus multiple neighbourhood structures, dynamic versus static objective function, and the difference between nature-inspired and non-nature inspired methods. A very similar categorisation is provided in [14]. The work in this thesis focuses on a subset of single-objective local search metaheuristics embedding *move acceptance methods* under a single point based search framework, an outline of which is given in Algorithm 1. In the scientific literature, some of such local search metaheuristics can also be referred to as *stochastic local search methods* [15], *reactive search methods* [16], *trajectory methods* [14] or *hyper-heuristics* [17]. From this point onward, we will refer to them solely as local search metaheuristics for consistency.

Algorithm 1: Outline of a Local Search Metaheuristic embedding Move Acceptance.

```
1  $s_0 \leftarrow generateInitialSolution()$;
2  $s_{best} \leftarrow s_0$;
3  $i = 0$;
4  $\mathbb{H} \leftarrow \emptyset$;
5  while $termination\ criteria\ not\ met$ do
6      $h \leftarrow random \in H$;
7      $s_i^{'} \leftarrow apply(h, s_i)$;
8      $(s_{i+1}, \mathbb{H}_{i+1}) \leftarrow acceptRejectDecision(f(s_i), f(s_i^{'}), i, \mathbb{H}_i)$;
9      if $f(s_i^{'}).isBetterThan(f(s_{best}))$ then
10         $s_{best} \leftarrow s_i^{'}$;
11     end
12     $i \leftarrow i + 1$;
13 end
14 return $s_{best}$;
```

A local search metaheuristic iteratively makes changes (perturbations) to a complete solution (s_i) at each iteration of the search (i) to produce a single neighbouring solution ($s_i^{'}$) by applying a random perturbation *move operator*, also referred to as a heuristic (h), to it (Line 7 of Algorithm 1). The resulting solution is then accepted or rejected for use as the solution in the next iteration (s_{i+1}) based on the decision of the embedded move acceptance method (Line 8 of Algorithm 1). The move acceptance method itself can maintain an internal history ($\mathbb{H}$), but can only do this by storing the *objective values* of previously visited solutions. The move acceptance method itself does not have access to domain, instance, or solution specific details.

Finally, when the algorithm terminates, the best solution found so far (s_{best}) is returned. The *objective* (evaluation, fitness, cost) *function* ($f(.)$) measures the quality of a solution and guides the search. The behaviour of a local search metaheuristic, and hence its ability to find good quality solutions, is determined by its neighbourhood and solution acceptance strategy, as defined by the move operators and its move acceptance method respectively.

A well-known weakness of local search is their tendency to get stuck in local optima. They therefore require some mechanism to counteract such situations during the search process, enabling the algorithm to explore other regions of the search landscape, potentially leading to solutions

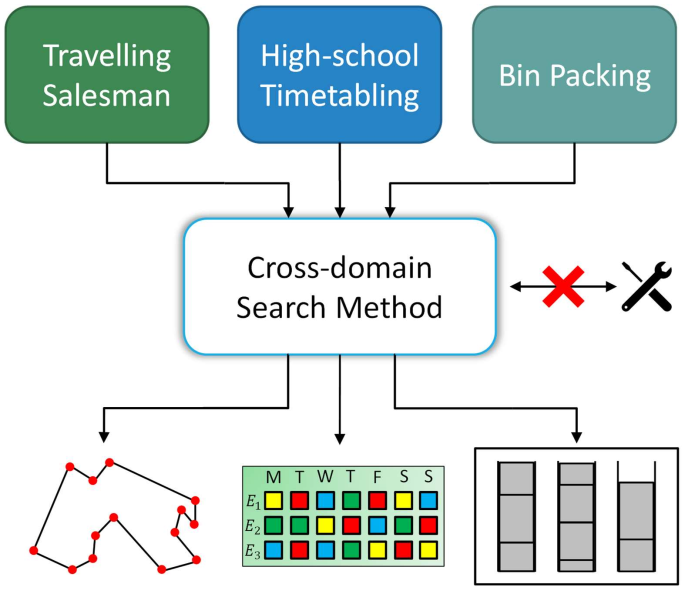


Figure 1.1: Illustration of a cross-domain search method solving different COPs without the need for expert intervention.

with better quality [15]. Hence, the use of a method accepting worse quality solutions as such a mechanism is a common strategy. There are many different previously proposed local search metaheuristics making use of a variety of move acceptance methods, such as Simulated Annealing [18], and Great Deluge [19].

Traditionally when practitioners are faced with solving a problem (even those from within the same domain), in addition to choosing suitable move operators and objective functions, they also have to choose a suitable move acceptance method to use alongside them in order for the overall algorithm to perform well. The state-of-the art in metaheuristic design is a trial-and-error process, involving tailoring the algorithmic components to the problem domain in hand, which are often not reusable when the same components are to be used for solving other problems [17]. For existing problems, the move acceptance method is usually chosen based on whichever worked well in the past, or which has received the most attention in the respective field. The choice of move acceptance method is, however, most often left unjustified. Moreover, the correct choice of move acceptance method becomes an even greater issue when the problem to be solved is *new* or *unknown* because the practitioner is then left with a dilemma for choosing the best move acceptance method for solving that problem without previous experience, knowledge, or guidance. An emerging area of research concerns the development of high-level, general-purpose search methodologies known as cross-domain search methods. An illustration of a cross-domain search method is shown in Figure 1.1 where a single cross domain search method is used to solve travelling salesman, high-school timetabling and bin packing problems without a need for any change in algorithmic components at the high level. Cross-domain search [20] is the term used to describe the high-level issue of devising a single search method which is able to solve multiple characteristically different COPs to a high quality given a pre-defined computational budget with the least, but preferably without, expert intervention or modification.

When a researcher or practitioner is tasked with solving a *new* or *unknown* (unseen) computationally hard COP, many questions arise concerning the decision-making process for choosing and designing the right search method. Three of the frequently asked questions are "Which algorithm should I use to solve my problem?", "What parameter settings/configuration should I use for my chosen algorithm?", and "Is there another algorithm which will produce better results than the one I have chosen?". The researchers and practitioners might have different goals for solving that unseen problem. Regardless of those goals, the decision-making questions can be combined into a single decision-making dilemma to "Is there a single algorithm that does not need to be re-tuned to perform at least as-good-as the existing algorithms that are themselves re-tuned for optimal performance to solve a given unseen problem?". Both practitioners and academics would benefit from answering that question. Using an off-the-shelf 'cheap' to apply and maintain algorithm, practitioners would save a lot of time and effort by not going through the standard trial-and-error process of developing a search method as a reasonable solution to their particular problem, requiring certain expertise. Such search methods would be of interest to many small to medium enterprises due to the high cost of developing tailored solutions. As for researchers, rather than customising an algorithm to solve the new problem, they can initially use the general-purpose search algorithm, again saving time and effort to form a benchmark of solutions for those instances of the new problem. Then they can investigate other tailored search methods using the results from the general-purpose search algorithm as basis for performance comparison to detect the best algorithm for the problem with the best configuration.

If an optimisation algorithm has a good cross-domain performance on a characteristically diverse set of known problems, then it can be reasonably expected to also perform well when solving *new* and *unknown* problems given that the algorithm is *not* a non-repeating algorithm [21] meaning that the search method never evaluates a search point more than once. Hence, the decision-making dilemma can be answered by using the current state-of-the-art cross-domain search method. As for the question "Is there another algorithm which will produce better results than the one I have chosen?", the answer will undoubtedly always be "yes" as researchers endeavour to produce higher and higher quality algorithms; but to obtain high-quality solutions to *new* or *unknown* problems at the present which are good enough in practise, and that are found in a reasonable time frame, the state-of-the-art cross-domain search method can be used.

COPs will perhaps always be solved better using exact methods or heuristic methods that are meticulously designed and re-designed for solving each problem and their variants. The issue however is that exact methods can sometimes fail to produce a solution in a reasonable time frame, and the financial investment and human cost of researching specialised problem solvers for every new problem can sometimes significantly outweigh that of an off-the-shelf (cross-domain search) algorithm which can be used as-is to produce solutions that are "good-enough".

Ever since the term "cross-domain search" was first used in 2011 [22], hyper-heuristics have been at the centre of attention in this area of research; perhaps in part because they were promoted as the solution to the cross-domain search problem as part of the Cross-domain Heuristic Search Challenge (CHeSC 2011). Hyper-heuristics have therefore emerged as solution methods to the cross-domain search problem and are formed of two components; a heuristic selection strategy, and a move acceptance criterion. Hyper-heuristic research has traditionally focused on investigating different heuristic selection strategies to improve their cross-domain performance, but this means that the move acceptance method is frequently neglected. The state-of-the-art hyper-heuristics combine some form of move acceptance method that is designed to inter-operate with a new heuristic selection strategy to achieve better cross-domain performance.

1.1 No Free Lunch Theorem

The No Free Lunch Theorem (NFLT) is a particular concern in the field of optimisation due to its implications on the design of algorithms for solving different problems. Broadly, the NFLT states that the performance of two algorithms when averaged over *all* cost functions is the same. At first glance, the NFLT appears to suggest that for all algorithms, there does not exist another algorithm that performs differently on average across all problems [21]. However, the original formulation of NFLT for optimisation discussed in [23] makes the following key assumptions:

- The search method is deterministic and non-repeating.
- The performance of each algorithm is evaluated across all cost functions.

Some heuristic search algorithms, such as those dealt with in this work, make use of neighbourhood/move operators that perform indiscriminate stochastic changes to the solution-in-hand to generate a candidate solution. That is, that candidate solutions are not chosen systematically from the set of unvisited candidate solutions, but rather generated stochastically to produce any candidate solution in the neighbourhood of the current solution; hence, it is entirely possible that an algorithm over time will re-evaluate and revisit the same candidate solutions previously visited. Without the use of additional data structures to remember each and every previously visited state, which is memory expensive, heuristic algorithms cannot guarantee that previous states are not revisited.

Another paper [24] discusses several definitions of the NFLT and the circumstances under which they cannot be applied to metaheuristics. The Sharpened NFLT extends the original NFLT by stating that the performance of algorithms are averaged across any set of objective functions which are closed under permutation (CUP).

The goal of cross-domain search methods is to design a single solution method which can be used 'off-the-shelf' to solve multiple *real world COPs.* These problems make use of different objective functions which are not contained in a single CUP set of objective functions as is the case in this study where the domains are limited to real-world COPs.

1.2 Research Motivation and Contributions

Research into selection hyper-heuristics as cross-domain search methods has typically focused on designing state-of-the-art heuristic selection strategies. Move acceptance methods have not been studied in-depth on their own before in the context of cross-domain search. A small scale study was performed in [25] comparing the performances of several move acceptance methods; however, that study used an adaptive large neighbourhood search framework and compares the move acceptance method to three problems, and not in the cross-domain context. This is despite it being suggested that the choice of move acceptance method has more effect on the cross-domain performance of a hyper-heuristic than that of a heuristic selection strategy in an initial study by [26] where a small number of simple low-level heuristics are applied for solving a set of benchmark function optimisation problems using binary representation.

The motivation for the work in this PhD comes from the fact that metaheuristics are sensitive to their parameter settings which leads to their performance for solving different COPs, and even instances from the same problem, to vary significantly. Parameter tuning is a common method used to improve the performance of metaheuristics for solving COPs. The very nature of metaheuristics means that their parameters are sometimes re-tuned even at the level of each instance for a variety of COPs such as the use of *instance-specific parameter tuning* of Guided Local Search for solving instances of the Travelling Salesman Problem [27] and *"empirical" tuning* of the parameters of Late Acceptance for solving instances of the Examination Timetabling problem [28]. It is stated in [29] that approximately 90% of the time required to design new heuristic methods and metaheuristics is expended on tuning their parameters. More recently, automated parameter tuning methods [30] have been developed which aim to produce a single parameter configuration that improves the performance of the search method. These tuning methods however are usually applied for multiple instances from the same problem domain, and hence would be re-ran when using the search method for solving different problems. These include *ParamILS* [31] which was used to tune the parameters of Simulated Annealing with Tabu Search for solving instances of the Quadratic Assignment Problem [32], and *irace* [33] which was used to tune the parameters of Iterated Local Search and an Iterated Greedy Algorithm for solving permutation and non-permutation flow shop problems [34].

Furthermore, different move acceptance methods are used for solving different COPs in the literature such as Simulated Annealing for solving Travelling Salesman problems [35], Late Acceptance for solving Examination Timetabling problems [28], and Record-to-Record Travel for solving heterogeneous fleet vehicle routing problems [36]. This highlights that different metaheuristics are more suited to solving specific problems and this is an issue for the design of a single general-purpose high-performance cross-domain search method.

Designing move acceptance methods that perform well across different COPs but independent of instance-specific and problem-specific features and parameter tuning efforts would provide a positive step forward for research into cross-domain search by contributing high-level strategies that have the potential to fulfil the aims of cross-domain search, and hence the decision-making dilemma - that is that a single solution method that can solve all COPs to a high quality, and without the need for expert intervention.

To highlight the point that the move acceptance methods used and designed in this work are targeted towards solving new and unknown problems, when a practitioner may want to find a quick and easy to obtain solution to a COP, and with only little knowledge or understanding of the domain, a (small subset of) simple perturbative operator(s) may only be implemented. In that case, the framework used in this work is ideal since it does not require the implementation of clever heuristics that exploit the properties and features of the underlying problem. This contrasts with the efforts of a researcher who will create many heuristics exploiting the problem features in which case solving such problems under a hyper-heuristic framework combining heuristic selection with move acceptance might outperform the single point-based stochastic local search metaheuristic method used in this study.

Restricting the scope of this work to investigate the effects of move acceptance on only local search metaheuristics has the benefit of eliminating potential confounding factors from related areas including, but not limited to, for example, population size and the strategy to maintain population diversity in population-based metaheuristics, the strategies to prohibit moves and tabu list length in Tabu Search, and the effects of heuristic selection within hyper-heuristics. The aim of this work is therefore to identify the various move acceptance methods used within *local search metaheuristics* as presented previously in Algorithm 1, investigate their cross-domain performance, and design a strategy for move acceptance that is able to improve the cross-domain performance of local search metaheuristics.

The contributions of the work carried out in this thesis are three-fold. First, in Chapter 2, a thorough survey on move acceptance methods within local search metaheuristics has been performed and a taxonomy is proposed for classifying move acceptance methods based on the characteristics of their mechanism for how the objective value of a candidate solution is used to determine whether it is accepted or rejected, and the nature of how their internal parameters are set. Secondly, in Chapter 4, the cross-domain performance of existing move acceptance methods, with one being chosen from each classification from the taxonomy, used under a local search metaheuristic framework is compared using an empirical study, and the effects of parameter tuning versus choice of the move acceptance method on the cross-domain performance of local search metaheuristics is explored. Finally, in Chapter 5, a novel move acceptance method (HAMSTA) is designed which has a cross-domain performance that is better than the existing methods under a single-point based stochastic local search metaheuristic framework despite HAMSTA using a single parameter configuration, requiring no expert intervention, for solving the given COPs whereas the existing methods utilise different parameter configurations for each problem domain.

1.3 Academic Output

The following lists itemises the articles and conference papers that were produced as a result of this research:

Journal Articles

- Warren G. Jackson, Ender Özcan and Robert I. John. **Move Acceptance in Local Search Metaheuristics for Cross-domain Search** *Expert Systems with Applications*, pp. 131-151, vol. 109, 2018.

Conference Papers

- Warren G. Jackson, Ender Özcan and Robert I. John. **Tuning a Simulated Annealing Metaheuristic for Cross-domain Search** *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1055-1062, 2017.

In Production

- Warren G. Jackson and Ender Özcan. **The History-based Adaptive Multi-stage Threshold Accepting Algorithm**.

1.4 Structure of Thesis

The structure of the remainder of this thesis is as follows. Chapter 2 provides a thorough survey on move acceptance methods within local search metaheuristics and a taxonomy is proposed for classifying local search metaheuristics based on the characteristics of their embedded move acceptance methods. An overview of the existing move acceptance methods is conducted with each method being classified based on the taxonomy. It is evident that there are a large number of move acceptance methods in the literature, and which have been used for solving many different COPs, either as components of metaheuristics or hyper-heuristics. Chapter 3 explains the local search metaheuristic framework used in this work alongside the problem domains, approaches to parameter tuning, move acceptance methods, methods of analysis, and the experimental designs. Chapter 4 provides a comparison of the cross-domain performance of existing move acceptance methods, with one being chosen from each classification from the taxonomy, used under a local search metaheuristic framework. From the investigation, which re-tunes the move acceptance methods for solving problems from different domains to show a best-case scenario of the move acceptance method's cross-domain performance, it is clear that some move acceptance methods perform better than others. Moreover, when comparing them based on the taxonomy, it is evident that adaptive move acceptance methods generally perform better than dynamic and static methods. Sometimes however, this is not always the case in the cross-domain context as some move acceptance methods have inherently poor designs which causes them to be inflexible for solving different problems, especially when the same parameter configuration is used. Furthermore, while increased parameter tuning efforts significantly improves the cross-domain performance of the local search metaheuristics, the choice of move acceptance method itself can have a significant effect on the cross-domain effectiveness of such algorithms. Therefore, rather than trying to overcome the parameter setting issue by designing a parameter tuning or control system for adapting the parameters of the existing move acceptance methods for cross-domain search, which would only lend an incremental improvement if the wrong acceptance is used, a novel move acceptance method is designed which attempts to perform as-good-as the existing move acceptance methods that are re-tuned for each domain being solved, but without itself requiring such expert intervention. Thus, eliminating the need for practitioners to have to decide which move acceptance they should use, and being able to be used off-the-shelf without the requirement to perform additional parameter tuning for each new problem they may come across for it to perform well. Researchers on the other hand are then able to use this move acceptance method in their own studies as a baseline target for comparison, enabling them to focus their time on developing specialised methods. Chapter 5 thus proposes a novel move acceptance method (HAMSTA) as a component of a local search metaheuristic for cross-domain search, and Chapter 6 concludes the thesis along with some future research directions being suggested which could potentially further improve the cross-domain performance of search algorithms based on the research into their move acceptance methods as conducted in this work.

1.5 Summary

Previous works in the area of cross-domain search have focused on improving heuristic selection strategies within hyper-heuristics. This is despite [26] suggesting that the choice of move acceptance method in a selection hyper-heuristic (as one classification of a cross-domain search method) has more effect on the performance of hyper-heuristics compared to their embedded heuristic selection method. In this work, a step back from the traditional research direction of improving heuristic selection in hyper-heuristics for cross-domain search is taken to gain an understanding of the influence of move acceptance. To be able to observe the effects of move acceptance specifically, a single-point based stochastic local search metaheuristic framework is used which is capable of abstracting away the confounding factors of heuristic selection strategies within hyper-heuristics and other related areas such as population-based search methods and tabu search. The investigation into the cross-domain performance of move acceptance methods in local search metaheuristics in the context of cross-domain search has not been done before, so the next chapter provides a critical review of such move acceptance methods and proposes a taxonomy for classifying them based on two of their design characteristics.

Chapter 2

Local Search Metaheuristics and Cross-domain Search

2.1 Introduction

This chapter, forming the literature review, contains an overview of related work on cross-domain search, a taxonomy for classifying move acceptance methods based on the natures of their internal parameter setting mechanisms and how the objective value of the candidate solution is compared to inform the accept/reject decision, and a comprehensive survey of the existing move acceptance methods. Cross-domain heuristic search is a term first used in [22] to describe the high-level issue of designing a single general-purpose heuristic search method which can be used to solve multiple characteristically different COPs to a high quality given a predefined computational budget and with the least, but preferably without, expert intervention. Since the first Cross-domain Heuristic Search Challenge in 2011, hyper-heuristics have emerged as methods for solving cross-domain search. Hyper-heuristics have existed in various forms since the 1960s [17], although the modern definition of hyper-heuristics as "heuristics to choose heuristics" was introduced forty years later [37]. Hyper-heuristics were then defined as a "search method or learning mechanism for selecting or generating heuristics to solve computational search problems" [17], and more recently in [38] as "*a hyper-heuristic is an automated methodology for selecting or generating heuristics to solve computational search problems.*". Selection hyper-heuristics are formed of two key components; a *heuristic selection strategy*, and a *move acceptance method*. It was suggested in [26] that the choice of move acceptance method in a selection hyper-heuristic has more effect on the performance of hyper-heuristics compared to the embedded heuristic selection method. Despite this, the research trend for cross-domain search since then has focused on improving the heuristic selection components, e.g. in [39, 40, 41]. In this thesis, the effects that move acceptance methods have on the cross-domain performance of a local search metaheuristic are therefore studied as this has not been done before with the emphasis on the context of cross-domain search. This thesis focuses on move acceptance methods as components of a subset of metaheuristics classified in [14] as single-point based metaheuristics with a static objective function, or as single-point trajectory methods with a static objective function in [42] - for consistency, such metaheuristics will be referred to as *local search metaheuristics* herein. The reason for focusing on move acceptance methods under a local search metaheuristic framework is to be able to observe the effects that *move acceptance* has on a cross-domain search method, as local search metaheuristics do not use any form of learning for selecting neighbourhood operators (as present in a hyper-heuristic framework), and as a single-point based search framework, does not require the configuration of the parameters of population-based metaheuristics such as population size and replacement strategy, and does not need to be concerned by tabu tenures in Tabu Search based methods and its strategies for exploring the neighbourhood of solutions. This focus allows for the reduction of as many experimental factors as possible, thus removing any possible confounding factors in this work; hence, the observations made in this work are due to the move acceptance methods themselves rather than any other decisions made within the experimental design.

The previous reviews on metaheuristics cover a variety of approaches, either conceptually, or in the context of specific problems [14, 43, 42, 12, 44, 45]. No distinction is made by the existing taxonomies for the embedded move acceptance method. In the remainder of this chapter, a taxonomy is therefore firstly provided for classifying move acceptance methods based on their characteristics in Section 2.3. A comprehensive survey of move acceptance methods from single-point based local search methods is performed in Section 2.4 with each move acceptance method also being classified based on the proposed taxonomy. The chapter is concluded in Section 2.5.

2.2 Related Work

In this section, the literature is discussed that relates to solving cross-domain search and papers that investigate the performances of move acceptance methods for solving combinatorial optimisation problems.

2.2.1 Methods for Solving Cross-domain Search

Cross-domain search has been tackled in the scientific literature mainly by the hyper-heuristic methods [17]. Hyper-heuristics were first described in [37] as *heuristics to choose heuristics.* Hyper-heuristics are high-level search methods which, unlike traditional methods such as meta-heuristics and genetic algorithms, search the space of low-level heuristics rather than the search space of solutions. Hyper-heuristics can therefore be applicable across different problem domains since the domain specific knowledge and expertise is embedded into their respective domains. There are two types of hyper-heuristics, selection hyper-heuristics where a heuristic selection mechanism chooses from a pre-defined set of low-level heuristics, and generation hyper-heuristics which use genetic programming to construct the heuristics [38]. Selection Hyper-heuristics have been previously decomposed into two key components; a heuristic selection mechanism, and a move acceptance criteria [26]. Under a traditional search framework, the heuristic selection mechanism is used to choose which heuristic(s) should be used to apply to the solution-in-hand to generate a candidate solution, determining the solution neighbourhood, and the move acceptance method is used to decide whether a candidate move is accepted or rejected, guiding the search. The operation of a selection hyper-heuristic iterates between these two components. Hyper-heuristics are also classified based on the learning mechanism used in the heuristic selection in [38] as either *online learning*, *offline learning* and *no learning.* A review of the selection and generation approaches are also discussed in [38]

Hyper-heuristics can be classified as single point-based or population-based. The move acceptance methods used in single point-based hyper-heuristics are reviewed in the following section. Here the population-based methods are discussed. A genetic and memetic algorithm (GA and MA) was used in a hyper-heuristic [46] for solving a trainer scheduling problem. [47] uses a GA for solving a Software Project Scheduling Problem using a hyper-heuristic framework. Steady-state Memetic Algorithms and Multimeme Memetic algorithms were used in a hyper-heuristic framework in [48] and were evaluated across six different COPs however the results showed that the performance of such population-based methods were outperformed by single-point approaches for cross-domain search.

2.2.2 Parameter Tuning for Cross-domain Search

While there has been significant efforts in the literature for devising automated parameter tuning methods, to date, there has only been one study which investigates parameter tuning with an emphasis on cross-domain search. A steady state memetic algorithm (SSMA) was tuned for cross-domain search in [49] where they used the Taguchi method to tune the population size and tournament size of the SSMA along with two domain-depend low-level heuristic parameters controlling the intensity of mutation and depth of search. The result of cross-domain parameter tuning revealed equal tour and population sizes, effectively turning SSMA into Iterated Local Search.

2.2.3 Comparisons of Move Acceptance Methods

There have been some studies in the literature where the performance of hyper-heuristics are evaluated using different move acceptance methods; however, such studies are limited with respect to cross-domain search.

A study [50] evaluated pairings of heuristic selection criteria and move acceptance methods; however, that study was limited to evaluating their performance across a variety of CVRP problem instances. What they found was that the best hyper-heuristic for solving all instances included the Naïve Acceptance strategy as the move acceptance method accepting all non-worsening moves, and worse moves with a fixed 50% probability.

Another study [25] compared the performances of several move acceptance methods; however, this was under an adaptive large neighbourhood search framework, and while they evaluate the performance of each method across three COPs, there was no emphasis or conclusions given on the general performance of each method across the three domains as a whole given a cross-domain search context. The findings of their work was that, in contrast to [50], Simulated Annealing performed the best for solving CVRP problems, and Record-to-record Travel performed the best for solving CMST and QAP problems when their parameters are re-tuned for each domain.

2.3 A Taxonomy for Move Acceptance Methods

In this section, a taxonomy is given for classifying move acceptance methods. While the work in this thesis focuses on a particular local search metaheuristic framework embedding move acceptance and stochastic perturbative move operators, this same taxonomy can be applied to other single-point based search frameworks that use such methods to determine the acceptance of a candidate solution. A classification for move acceptance methods is shown in Figure 2.1 and distinguishes them based on two features as indicated by the dashed arrows. Firstly (left), the nature of the accept/reject decision, as indicated by the *acceptRejectDecision()* procedure in Line 8 of Algorithm 1, which can be found in Chapter 1, is considered which takes into account what type of value the objective

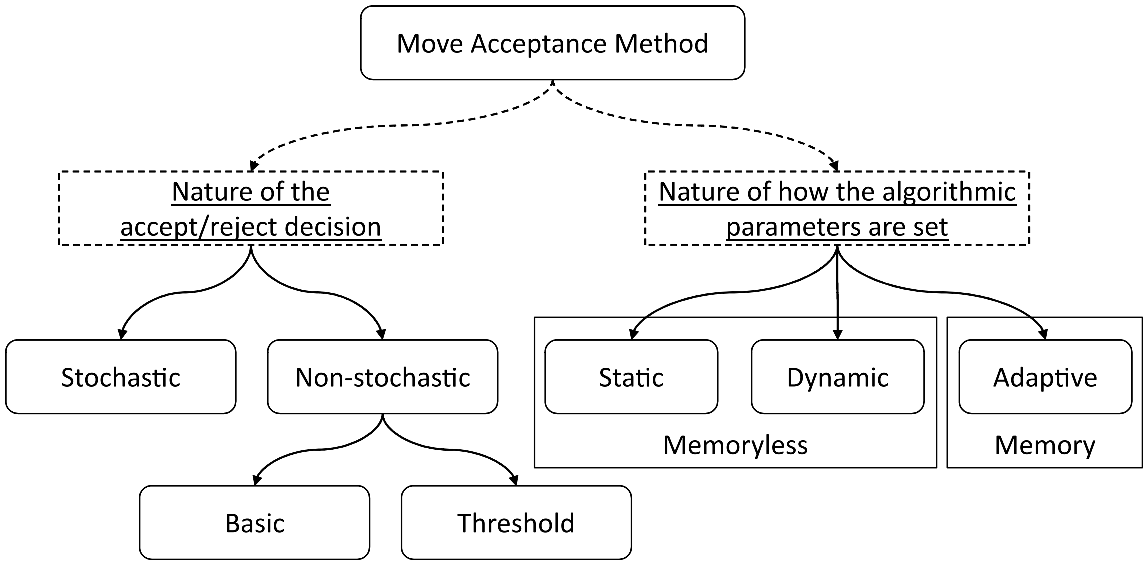


Figure 2.1: A taxonomy for move acceptance methods based on the natures of the accept/reject mechanism and how the internal algorithmic parameters are set.

value of the candidate solution is compared to, and the resulting probability of acceptance that is returned. The second part of the classification (right) considers the nature of how the internal algorithmic parameters are set within the move acceptance method. The mechanism(s), if any, to update the settings of the internal parameters are employed within the move acceptance methods themselves; hence, this taxonomy can also be applied to move acceptance methods within other local search frameworks.

2.3.1 Classification of the Accept/Reject Decision

The first part of the classification concerns whether the acceptance of a solution in any given move is deterministic or not. There are two fundamental mechanisms for accepting or rejecting non-improving solutions. On one hand, the acceptance mechanism accepts or rejects a solution outright by using a binary operator to compare the objective value of the candidate solution to some acceptance threshold value. These methods are what we call as *non-stochastic move acceptance methods.* On the other hand, the acceptance mechanism can accept a solution probabilistically, either directly by assigning a probability that the move acceptance should accept a solution, or indirectly by subjecting the parameter(s) affecting the value of the acceptance threshold to randomness. These methods are what we call as *stochastic move acceptance methods.*

There are two strategies used to calculate the values used for the acceptance threshold in non-stochastic basic move acceptance methods. The taxonomy therefore includes a secondary level to further classify such methods. The first strategy, called as *non-stochastic basic*, reuses the objective values of previously encountered solutions for the accept/reject decisions. These either use the objective value of the current solution, or maintain a memory of objective values of visited solutions to decide on an appropriate value for the acceptance threshold at a given point during the search. The second strategy on the other hand, called as *non-stochastic threshold*, uses any real value as the value of the acceptance threshold. The values of the acceptance threshold are therefore not guaranteed to correspond to the objective value of any solution to the problem being solved.

The overall classification of a move acceptance method based on the nature of the accept/reject decision follows a trivial precedence. If the mechanism deciding whether to accept or reject a solution uses, or is influenced by parameters utilising, stochastic nature then it is *stochastic*, irrespective of the acceptance threshold. If the acceptance mechanism is non-stochastic but uses acceptance thresholds during any point of the search which are not objective values of previous solutions, then it is *non-stochastic threshold*. Otherwise, if the acceptance mechanism only uses objective values of previous solutions for the acceptance threshold, then it is *non-stochastic basic*. This can be summarised by the following precedence order relationship ($x \prec y : x$ precedes y).

$$\text{stochastic} \prec \text{non-stochastic threshold} \prec \text{non-stochastic basic}$$

2.3.2 Classification of the Algorithmic Parameter Setting

The second part to the classification concerns the nature of the parameter settings and the mechanisms used to control them. These are broken down into three classifications, *static*, *dynamic*, and *adaptive*. A similar classification exists for parameter control in the context of multi-point search methods, namely evolutionary algorithms (EAs) such as those put forward in [51, 52]. Our taxonomy abstracts upon those classifications by only considering whether parameter tuning is based on what they refer to as a non-iterative control method (static) or an iterative control method (adaptive). Moreover, ours extends their view on iterative control methods by considering whether an algorithmic parameter setting mechanism relies on the overall number of iterations/time budget, along with the current step/elapsed time (dynamic), and/or acts upon search history (adaptive).

If a move acceptance method has multiple algorithmic parameter settings, then the final classification of the move acceptance method based on the nature of the parameter settings and the mechanisms used to control them is determined by the following precedence relation:

$$\text{adaptive} \prec \text{dynamic} \prec \text{static}$$

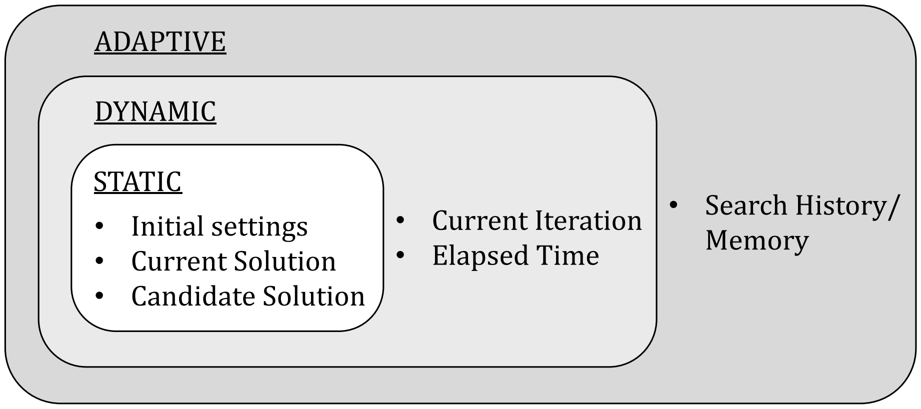


Figure 2.2: Venn diagram illustrating the information used by each algorithmic parameter setting mechanism to decide on the settings of the parameters of the move acceptance method.

The overall classification based on the algorithmic parameter setting mechanisms can be determined using the descriptions below. This can be encapsulated as a Venn diagram as shown in Figure 2.2.

Static algorithmic parameter setting mechanisms A move acceptance method is classified as having a *static* nature of how the algorithmic parameters are set if given the same candidate and current solutions, the acceptance threshold or acceptance probability would be the same irrespective of the current elapsed time or iteration count and search history.

Dynamic algorithmic parameter setting mechanisms A move acceptance method is classified as having a *dynamic* nature of how the algorithmic parameters are set if given the same candidate and current solutions at the same current elapsed time or iteration count, the acceptance threshold or acceptance probability would be the same irrespective of search history.

Adaptive algorithmic parameter setting mechanisms A move acceptance method is classified as having an *adaptive* nature of how the algorithmic parameters are set if given the same candidate and current solutions at the same current elapsed time or iteration count, the acceptance threshold or acceptance probability is not guaranteed to be the same as one or more components depend on search history.

2.3.3 Example Classification using Simulated Annealing

Here we provide a breakdown of the classification of Simulated Annealing (SA), a very popular move acceptance method employed in local search metaheuristics. Starting with the nature of the accept/reject mechanism, Simulated Annealing would be classified as having a *stochastic* accept/reject mechanism. Despite accepting non-worsening moves outright, worse quality moves are accepted based on an acceptance probability determined by the Metropolis criterion. All non-reheating cooling schedules, such as geometric cooling and Lundy and Mees's cooling, contain mechanisms to reduce an internal temperature setting over *time*. In some cases, these mechanisms are not affected by search history and depend only on the elapsed duration of the search, although various implementations exist which, for example, reduce this temperature after a certain number of accepted moves. Such mechanisms are therefore either *dynamic* or *adaptive* depending on their implementations. All other parameters, such as α for geometric cooling, and β for Lundy and Mees's cooling, remain fixed, hence these are *static*. The acceptance probability, given an elapsed duration, will therefore always be the same given the same current and candidate solution objective values. Hence, the classification for the nature of Simulated Annealing's algorithmic parameter setting mechanisms is *dynamic*. In conclusion, Simulated Annealing, depending on its particular implementation can either be a *dynamic stochastic* move acceptance method, or *adaptive stochastic* move acceptance method.

2.4 A Survey of Existing Methods

Local search metaheuristics are composed of two main components; a neighbourhood structure as defined by the move operators, and a move acceptance method which is used to guide the search. A survey of the move acceptance methods that have been used for solving various single-objective COPs are discussed below including the targeted problems and the parameter configurations that were used by the authors for solving each problem. Some of these move acceptance methods were used within a hyper-heuristics that embeds multiple move acceptance methods in a single execution and are denoted by the subscript ^{M}. This is the case for all multi-stage and genetic programming-based hyper-heuristics. The review is restricted to those move acceptance methods used as components of single point-based heuristic search methods, both those used for solving a targeted problem, and those with a focus on solving multiple problems (cross-domain). Hyper-heuristics were proposed as the solution method for solving the cross-domain search problem and the state-of-the-art cross-domain search methods are hyper-heuristics. Hence, it would be interesting to see if the move acceptance methods that are utilised within hyper-heuristics may provide some clues/information as to which move acceptance strategy(ies) perform well, theoretically or in practice, for solving the cross-domain search problem.

Table 2.16 forms a succinct overview of the existing move acceptance methods from the literature and classifies them based on the taxonomy given in Section 2.3. The abbreviations used for each move acceptance method are summarised in Table 2.17.

2.4.1 All Moves (AM)

While in the literature, all moves [37] is used as an 'acceptance' strategy, it in fact does nothing to restrict any move from being accepted and is not really a move acceptance method. The review of AM is therefore only included for completeness. AM accepts all moves, unconditional of the candidate solutions objective value, as shown in Equation 2.1. AM is parameter-free meaning that it does not contain any parameters; hence, the same configuration is, by definition, used for solving all problems. AM has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.1. In fact, other move acceptance methods classified as non-stochastic threshold and stochastic can be transformed into AM by manipulating their parameter settings to extreme values. For example, Threshold Accepting 2.4.6 could be used with a very high setting for *tau*, Great Deluge 2.4.12 can have a very high target value (though some moves at the beginning may be restricted if not set high enough), and Simulated Annealing 2.4.19 given sufficiently high start and end temperature settings.)

$$s_{i+1} \leftarrow s_i^{'} \tag{2.1}$$

While AM is primitive and perhaps naïve, it should be noted that the papers accepting all moves do so with the intent of observing the effects of a proposed heuristic selection strategy under a selection hyper-heuristic framework, hence its apparent popularity.

2.4.2 Only Improving (OI)

Only Improving acceptance [37], sometimes referred to as Improving Only, is a *static non-stochastic basic* move acceptance method which only accepts moves of improving quality, as shown in Equation 2.2.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) < f(s_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.2}$$

OI is parameter-free meaning that it does not contain any parameters; hence, the same config-

Table 2.1: COPs solved using All Moves (AM) move acceptance.

Problem Domain (COP)	**Source(s)**
Benchmark function optimisation	[53] [26] [54]
Capacitated Vehicle Routing Problem	[55]
Channel assignment problem	[56]
Component Placement Sequencing For Multi Head Placement Machine	[57]
Eternity II Puzzle	[58]
Exam Timetabling	[53] [55]
Generalised Assignment Problem	[59]
High-school timetabling	[60]
Maintenance scheduling	[61]
Nurse rostering	[62] [63]
P-Median Problem	[64]
Permutation flow shop	[65]
Personnel scheduling	[63]
Project presentation scheduling	[63] [66] [67]
Sales Summit Scheduling	[68] [37] [63]
Shelf Space Allocation	[69]
Short-term electrical power generation scheduling problem	[70]
Sports Scheduling	[71]
University Course Timetabling	[62]
Cross Domain Search (CHeSC 2011 domains)	[72] [39] [73] [74] [75] [76]^M [41]^M

uration is, by definition, used for solving all problems. OI has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.2.

Table 2.2: COPs solved using Only Improving (OI) move acceptance.

Problem Domain (COP)	**Source(s)**
Benchmark function optimisation	[53]
	[26]
	[54]
Capacitated Vehicle Routing Problem	[55]
Channel assignment problem	[56]
Component Placement Sequencing For Multi Head Placement Machine	[57]
Exam Timetabling	[53]
	[77]
	[55]
Generalised assignment problem	[59]
High-school timetabling	[60]
Maximum satisfiability	[78]
Multidimensional 0-1 knapsack problem	[79]
	[80]
Nurse rostering problem	[63]
	[81]
One-dimensional bin packing	[78]
P-Median Problem	[64]
Patient admission scheduling problem	[81]
Permutation flow shop	[65]
	[78]
Personnel scheduling	[63]
	[78]
Project presentation scheduling	[67]
	[63]
	[66]
Sales Summit Scheduling	[37]
	[68]
	[63]
Shelf Space Allocation	[69]
Short-term electrical power generation scheduling problem	[70]
Unit Commitment Problem	[82]
University course timetabling	[77]
Cross Domain Search (CHeSC 2011 domains)	[41]^{M}

2.4.3 Improving or Equals (IE)

Improving or equals acceptance is a *static non-stochastic basic* move acceptance method which accepts moves of non-worsening quality only, as shown in Equation 2.3.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.3}$$

IE is parameter-free; hence, the same configuration is, by definition, used for solving all problems. IE has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.3.

Table 2.3: COPs solved using Improving or Equals (IE) move acceptance.

Problem Domain (COP)	**Source(s)**
Benchmark function optimisation	[53]
	[26]
	[83][1]
Eternity II Puzzle	[58]
Exam Timetabling	[53]
	[28]
	[84]
Home care scheduling	[85]
Nurse rostering problem	[81]
Patient admission scheduling problem	[81]
Personnel routing and rostering	[86]
Ready-mixed concrete delivery problem	[87]
Short-term electrical power generation scheduling problem	[70]
Sports Scheduling	[71]
Cross Domain Search (CHeSC 2011 domains)	[88]
	[73]
	[74]
	[40]

2.4.4 Naïve Acceptance (NA)

Naïve Acceptance [89] is a *static stochastic* move acceptance method which accepts all non-worsening moves, and worse moves based on a fixed probability which is by definition set to 50%, as shown in Equation 2.4.

$$s_{i+1} \leftarrow \begin{cases} s_i' & f\left(s_i'\right) \leq f\left(s_i\right) \vee random \in [0,1] < 0.5 \\ s_i & \text{otherwise} \end{cases} \tag{2.4}$$

NA has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.4. Although any percentage could be used for naïvely

[1]IE acceptance was used alongside a greedy solution acceptance strategy; hence, is in a strict sense, it is *greedy with IE acceptance*.

accepting worse moves, this was set as 50% (as shown as 0.5 in Equation 2.4) by all [89, 90, 55, 74, 40, 41].

Table 2.4: COPs solved using Naïve Acceptance (NA) move acceptance.

Problem Domain (COP)	**Source(s)**	P
Capacitated Vehicle Routing Problem	[55]	0.5
Exam Timetabling	[55]	0.5
Maximum satisfiability	[90]	0.5
One-dimensional Bin Packing	[89]	0.5
	[90]	0.5
Permutation Flow Shop	[89]	0.5
	[90]	0.5
Personnel Scheduling	[89]	0.5
	[90]	0.5
Cross Domain Search (CHeSC 2011 domains)	[74]	0.5
	[40]	0.5
	[41]^M	0.5

2.4.5 Adaptive Acceptance (AA)

Adaptive Acceptance [89] is an *adaptive stochastic* move acceptance method which accepts all non-worsening moves, and accepts worse moves based on a probability, *acceptanceRate*, which is initially set to 0%, but increased (decreased) by a fixed amount for every period of time that has elapsed without improving the solution (without generating worse moves). The mechanism for accepting a move using AA is shown in Equation 2.5 where *acceptanceRate* is controlled adaptively based on "*whether the search is thought to be progressing or stuck in a local optimum*" [89] and as discussed below. This is different to Simulated Annealing, see Section 2.4.19, where the 'acceptance rate is determined based on the move delta and system temperature as controlled by an internal cooling schedule.'. AA has been used for solving a variety of COPs such as those shown in Table 2.5.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee random \in [0,1] < acceptanceRate \\ s_i & \texttt{otherwise} \end{cases} \tag{2.5}$$

Table 2.5: COPs solved using Adaptive Acceptance (AA) move acceptance.

Problem Domain (COP)	**Source(s)**
Capacitated Vehicle Routing Problem	[55]
Exam Timetabling	[55]
One-dimensional Bin Packing	[89]
Permutation Flow Shop	[89]
Personnel Scheduling	[89]

The increment was set to 5% with a time period of 0.1 seconds in [89]. The same 5% increment was used in [55] alongside an unspecified number of consecutive non-improving iterations for increasing *acceptanceRate*, and reducing *acceptanceRate* whenever a solution is *accepted*.

2.4.6 Threshold Accepting (TA)

Threshold Accepting [91] is originally a *dynamic non-stochastic threshold* move acceptance method which accepts all moves whose solution value is not worse than a threshold value which is calculated as the sum of the current solution, and a non-negative offset, T, as shown in Equation 2.6.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) + T \\ s_i & \texttt{otherwise} \end{cases} \tag{2.6}$$

A single parameter, T, controls the acceptance threshold. Two definitions of TA exist in the literature which spans the *static* and *dynamic* “nature of how the algorithmic parameters are set” classifications, as used in [55] and [91] respectively. Both definitions have a non-stochastic threshold “nature of the accept/reject decision”. Note that while [91] proposed TA, they do not use it as a component of a hyper-heuristic framework but is built around the problem instance(s) being solved, and therefore their method for updating T cannot be reasonably translated as a move acceptance component for cross-domain search. TA was used in [55] as the move acceptance method of a hyper-heuristic for solving both the Capacitated vehicle routing and Exam timetabling problems where T was fixed as 0.03.

2.4.7 Backtracking Adaptive Threshold Accepting (BATA)

Backtracking Adaptive Threshold Accepting [5] is an *adaptive non-stochastic threshold* move acceptance method which accepts all non-worsening moves, and moves whose objective value does not exceed a threshold value equal to the sum of the current solution, and a parameter τ such that $\tau > 0$. τ is reduced such that it is decreased after a fixed number of iterations $iters$, but is increased (“backtracked”) to a value close to the previous threshold value if no solution is accepted during the previous $iters$ iterations, as shown in Equation 2.7.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) + \tau \\ s_i & \texttt{otherwise} \end{cases} \tag{2.7}$$

BATA was used in [5] under a random choice hyper-heuristic framework for solving the Heterogeneous fixed fleet vehicle routing problem, and they found that BATA performs at its best if the backtracked threshold setting is still less than the previous setting. The initial value of τ was empirically determined for each problem instance in the range $[11, 76]$ such that the settings were different for each instance; however, the setting for T_b remained at the same 0.98 value for all problem instances.

2.4.8 Step Counting Hill Climbing (SCHC)

Step Counting Hill Climbing [92] is an *adaptive basic* move acceptance method which is a novel extension of one of the most basic move acceptance methods - IE (see Section 2.4.3). In SCHC, non-worsening moves are always accepted, and worse moves are accepted if the objective value of the candidate solution is (strictly) less than a threshold boundary equal to the objective value of the current steps initial solution, where each step lasts for a number of iterations, as defined by the counting strategy. The acceptance mechanism of SCHC is shown in Equation 2.8 where B is the objective value of the initial solution of the current stage whose length is determined based on various strategies as discussed below.

$$s_{i+1} \leftarrow \begin{cases} s_i' & f\left(s_i'\right) \leq f\left(s_i\right) \vee f(s_i) < B \\ s_i & \text{otherwise} \end{cases} \tag{2.8}$$

SCHC can use different counting strategies to adaptively determine the length of each “step”. [92] investigated three of such strategies called as; SCHC-*acp*, SCHC-*imp*, and SCHC-*all*. These strategies count the number of; accepted moves, improving moves, and all moves respectively. Once the counting strategy exceeds a target value, L_c, the acceptance threshold, B, is updated as the objective value of the current solution and the counter is reset.

SCHC was used in [92] under a random choice hyper-heuristic framework for solving the University course timetabling problem and set $L_c = 200$ for solving all problem instances. Furthermore, they investigated the effect of varying this parameter on the performance of SCHC with L_c being chosen randomly from the range $[0, 65000]$.

2.4.9 Iteration Limited Threshold Accepting (ILTA)

Iteration Limited Threshold Accepting [93] is an *adaptive non-stochastic threshold* move acceptance method which accepts all non-worsening moves, and accepts worsening moves, after a consecutive number of worse candidate moves are generated, whose objective value does not exceed a threshold value calculated as a factor of the objective value of the best solution found so far. The acceptance strategy for ILTA is shown in Equation 2.9, where ϵ is fixed to a pre-defined value that is used to calculate a threshold value based on the objective value of the best solution found so far. *w_iterations* is a counter which is updated at each iteration to track the current number of consecutively generated worse moves, and k is a fixed value which is defined to control when the threshold part of the acceptance strategy is used by comparison with *w_iterations*. That is, when the number of consecutive number of worse generated moves (*w_iterations*) exceeds an upper limit (k), the threshold based acceptance strategy is enabled.

Note that the original acceptance strategy only worked for problem domains with objective functions that returned positive values; hence, the version displayed here has been corrected for use in all objective value ranges.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee \\ & w\_iterations \geq k \wedge f(s_i^{'}) \leq f(s_{best}) + |f(s_{best}) \times \epsilon| \\ s_i & \texttt{otherwise} \end{cases} \tag{2.9}$$

ILTA was used in [93], [85], and [58] as the move acceptance method of a hyper-heuristic for solving the Travelling tournament problem, Home care scheduling problem, and Eternity II puzzle respectively. [93] set ϵ depending on the instance being solved and associated time limit with the smallest setting being 0.01 and the largest setting being 0.20. Larger settings were used for those instances with shorter computational budgets whereas smaller settings were used for those instances with longer computational budgets. Despite some instances having the same computational budget, their settings for ϵ were not always the same. k was fixed as 100 for all problem instances. [85] set ϵ as both 0.003 and 0.004 during separate evaluations of all instances, and fixed $k = 100$ for all problem instances. [58] used an ϵ value of 0.4 for solving the Eternity II puzzle, and with k fixed to 500 iterations for all problem instances. This is despite the instances varying in size, and the larger instances having a larger (time-based) computational budget.

2.4.10 Adaptive Iteration Limited Threshold Accepting (AILTA)

Adaptive Iteration Limited Threshold Accepting [85] is an *adaptive non-stochastic threshold* move acceptance method which accepts all non-worsening moves, and accepts worsening moves whose objective value does not exceed a threshold value calculated as an adaptively updated factor of the cost of the best solution found so far, as shown in Equation 2.10, where ϵ is chosen from an ordered list of pre-defined real values such that the value of ϵ is increased over time as AILTA fails to improve the best solution found. *w_iterations* and k are both control parameters for enabling/disabling the threshold part of the acceptance strategy and are the same as when used in ILTA (described above in Section 2.4.9).

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee \\ & w\_iterations \geq k \wedge f(s_i^{'}) \leq f(s_{best}) + |f(s_{best}) \times \epsilon| \\ s_i & \texttt{otherwise} \end{cases} \tag{2.10}$$

AILTA was used in [85] as the move acceptance method of a hyper-heuristic for solving the Home care scheduling problem where ϵ was incremented after a fixed 5000 consecutive moves where the best solution is not improved, and they investigated using different sets of values for ϵ such that either:

1. $\epsilon \in \{1.003, 1.004, 1.005, 1.006, 1.007, 1.008, 1.009, 1.010\}$
2. $\epsilon \in \{1.003, 1.004, 1.005, 1.006, 1.007\}$
3. $\epsilon \in \{1.004, 1.005, 1.006, 1.007\}$

In all cases, the values of ϵ increase by 0.001 each time it is incremented. The performance of AILTA with the different sets of parameter values varied depending on the instance of the HCSP that was being solved. As instance sizes decreased, there was no clear correlation between the parameter value sets and performance of AILTA. AILTA with $\epsilon \in$ {1.003, 1.004, 1.005, 1.006, 1.007, 1.008, 1.009, 1.010} performed the best on the two largest, and two smallest HCSP instances, whereas $\epsilon \in$ {1.004, 1.005, 1.006, 1.007} allowed AILTA to perform better on the two intermediate sized instances.

2.4.11 Record-to-record Travel (RRT)

Record-to-record Travel [19] is an *adaptive non-stochastic threshold* move acceptance method which accepts all non-worsening moves, and accepts worse moves whose candidate solution's objective value is not worse than a threshold value equal to the cost of the best solution found so far $f(s_{best})$ added to an offset parameter, known as the `DEVIATION` parameter, as shown in Equation 2.11.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_{best}) + \texttt{DEVIATION}\ \textit{where}\ \texttt{DEVIATION} > 0 \\ s_i & \texttt{otherwise} \end{cases} \tag{2.11}$$

RRT has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.6.

Table 2.6: COPs solved using Record-to-record Travel (RRT) move acceptance with the settings used for `DEVIATION`.

Problem Domain (COP)	**Source(s)**	`DEVIATION`
Channel assignment problem	[56]	2
High-school timetabling	[60]	0.5
Cross Domain Search (CHeSC 2011 domains)	[76]^{M}	30 `OR` 30 + 'decimal' (adaptive)
	[94]	$\epsilon \times f(s_{best})$

RRT was used as the move acceptance method of a hyper-heuristic for solving a single problem in [56] and [60] where `DEVIATION` was fixed equal to 2 and 0.5 respectively. [76] and [94] used RRT as the move acceptance method of a hyper-heuristic for solving the cross-domain search problem (CHeSC domains) where [76]^{M} used a hidden Markov model for finding and associating sequences of heuristics with either RRT or AM move acceptance methods, and set `DEVIATION` proportional to the objective function value such that if $f(s_i^{'}) > 0$, then `DEVIATION` $= 30$, otherwise, `DEVIATION` is set such that *"30 is added to the decimal part of the objective value"* [76]. Moreover, the `DEVIATION` setting is reduced by 2 upon successive periods (15 seconds) of non-improvement. [94] proposed a stage-based variant of RRT where instead of calculating the acceptance threshold value from the best solution found so far, the objective value of the best solution found during the current stage is used according to `DEVIATION` $= \epsilon \times f(s_{best})$ where ϵ is calculated as $\frac{\lfloor log(f(s_{best}))\rfloor + c_i}{f(s_{best})}$ such that c_i is a circular list of values containing the objective values of a number of best solutions found.

2.4.12 Great Deluge (GD)

Great Deluge [19] is a *dynamic non-stochastic threshold* move acceptance method which accepts all moves whose solution value is either not worse than that of the current solution, or not worse than a threshold value, τ_i, which is linearly decreased over time from an initial value to a target value as shown in Equation 2.12.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq \max(f(s_i), \tau_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.12}$$

The threshold value, τ_i, can be calculated such that is linearly decreased over time from an initial threshold value, τ_0, to a value equal to the lower bound, *qualityLB*. The time-based calculation is given in Equation 2.13 whereas the iteration-based calculation is given in Equation 2.14 such that i

is the current iteration, and n is the maximum iterations as defined by the computational budget.

$$\tau_i = qualityLB + (\tau_0 + qualityLB) \times \left(1 - \frac{T_{elapsed}}{T_{total}}\right) \tag{2.13}$$

$$\tau_i = qualityLB + \tau_0 \times \left(1 - \frac{i}{n}\right) \tag{2.14}$$

In the literature, the initial threshold value is exclusively set to be equal to the objective value of the current solution, $\tau_0 = f(s_0)$, however multiple strategies exist for determining the final threshold value. Moreover, multiple control methods for updating the threshold value are present in the literature, and these form distinct Great Deluge variants, Extended Great Deluge and Flex Deluge, and these are covered in their respective sections 2.4.15 and 2.4.13.

There is also an *adaptive non-stochastic threshold* definition of GD, as used in [95] which considers the objective value of the candidate solution to update the threshold value at each step, and the threshold value is decreased exponentially over time, rather than the traditional and more widely used linear decrease rate explained previously. GD has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.7.

Table 2.7: COPs solved using Great Deluge (GD) move acceptance with the parameter settings for the initial threshold value τ_0 and final threshold values $qualityLB$.

Problem Domain (COP)	**Source(s)**	τ_0	$qualityLB$
Benchmark function optimisation	[53]	$f(s_0)$	Not stated
	[26]	$f(s_0)$	0
	[84]	$f(s_0)$	Not stated
Capacitated Vehicle Routing Problem	[55]	$f(s_0)$	‘estimated lower bound’
Channel assignment problem	[96]	$f(s_0)$	‘best solution cost from literature’
Eternity II Puzzle	[58]	$f(s_0)$	Not stated
Exam timetabling	[53]	$f(s_0)$	Not stated
	[97]	$f(s_0)$	‘lower bound of objective function’
	[55]	$f(s_0)$	‘estimated lower bound’
Grid Scheduling	[98]	$f(s_0)$	Not stated
High-school timetabling	[60]	$f(s_0)$	Not stated
Nurse rostering problem	[81]	$f(s_0)$	0
One-dimensional bin packing	[89]	$f(s_0)$	‘expected final objective value’
Patient admission scheduling problem	[81]	$f(s_0)$	0
Permutation flow shop	[89]	$f(s_0)$	‘expected final objective value’
Personnel routing and rostering	[86]	$f(s_0)$	0
Personnel scheduling	[89]	$f(s_0)$	Not stated
Ready-mixed concrete delivery problem	[87]	$f(s_0)$	0
Short-term electrical power generation scheduling problem	[70]	$f(s_0)$	Not stated
Sports scheduling	[71]	$f(s_0)$	Not stated
University course timetabling	[92]	$f(s_0)$	‘varied randomly’
Cross Domain Search (CHeSC 2011 domains)	[88]	$f(s_0)$	Not stated

The target threshold value for Great Deluge, *qualityLB*, from papers solving problems from a single domain set *qualityLB* as either zero [84, 87, 86], the objective value of the best known solution from the literature [96], or the lower bound of the objective function [97]. [92] randomly varied *qualityLB* as an investigation of a few move acceptance method's performance for solving the targeted problem. Those papers which solve more than one problem, but not as cross-domain search set *qualityLB* as either the expected final objective value/estimated lower bound [89, 55], or zero [81] for all domains. [88] applied GD for solving the cross-domain search problem but they did not specify the parameter settings that were used. The parameter configurations were either not stated, or said to be empirically determined, in [53, 98, 26, 71, 58, 60].

2.4.13 Flex Deluge (FD)

Flex Deluge [99] is a potential *adaptive non-stochastic threshold* move acceptance method that accepts all non-worsening moves, and worse moves whose objective value is not worse than a threshold value. FD is similar to Great Deluge (see Section 2.4.12), but where the acceptance threshold value is adjusted between the current threshold value and the objective value of the current solution using a coefficient k_f such that $0 \leq k_f \leq 1$ as shown in Equation 2.15.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \ \wedge \ f(s_i^{'}) \geq \tau_i \ \vee \ f(s_i^{'}) \leq f(s_i) + k_f(\tau_i - f(s_i)) \ \wedge \ f(s_i^{'}) \leq \tau_i \\ s_i & \texttt{otherwise} \end{cases} \tag{2.15}$$

The mechanism for controlling τ is identical to that used in the original Great Deluge and is therefore covered in Section 2.4.12. FD has been used in both [99] and [100] for solving the Exam timetabling problem. The settings for τ_0 and *qualityLB*, where stated, are all in line with the settings that were used for the standard Great Deluge algorithm with both [99] and [100] using the objective value of the initial solution ($f(s_0)$) as the initial threshold value for all problem instances, and [100] using a setting of zero for the target threshold value (*qualityLB*) for all problem instances. No setting was given for the target threshold value in [99].

While FD was proposed in [99], no mechanism was given for how to control k_f other than stating that *"By varying k_f, it is possible to obtain an algorithm with characteristics of both the original Great Deluge (k_f = 1) and Greedy Hill-Climbing (k_f = 0).".* It can only be presumed that the addition of a coefficient parameter was meant to be adaptively controlled based on the search state - hence FD was stated above to be potentially adaptive. In contrast, [100] fixed the coefficient value to 0.5 for all problem instances constituting a *dynamic non-stochastic threshold* variant of FD.

2.4.14 Non-linear Great Deluge (NLGD)

Non-linear Great Deluge [1] is an *adaptive stochastic* move acceptance method which accepts all non-worsening moves, and worse moves that do not exceed a threshold value which is generally decreased at a non-linear rate as shown in Equation 2.16. While NLGD initially appears as an adaptive non-stochastic threshold move acceptance method, it is actually classified as adaptive *stochastic* due to the use of a random number generator which is used to determine the rate at which the threshold, B, is reduced at each step.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq \max(f(s_i), B_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.16}$$

The threshold value, B, is reduced non-linearly over time by multiplying the current setting with a randomised value. In addition, NLGD contains a strategy to increase the value of B whenever the search is thought to be about to converge. The iteration-based calculation is given below such that β is set to the target threshold value, and min and max are lower and upper bounds which determine the rate at which B decreases, and B_{min} and B_{max} determine upper and lower bounds for a random value that is added on to B when the search is thought to be stuck. $isLargeOrSmallInstance$ is a problem specific variable which signals to NLGD whether the problem being solved is a small instance or large instance, but not a medium sized instance. f_{low} is a value which is used in an attempt to detect whether the search state is stuck in a local optima such that B can be increased.

$$B_{i+1} \leftarrow \begin{cases} B_i + random \in [B_{min}, B_{max}] & B_i - f(s_i^{'}) < 1 \wedge \\ & \quad (isLargeOrSmallInstance \vee f(s_{best}) < f_{low}) \\ B_i + 2 & B_i - f(s_i^{'}) < 1 \wedge \\ & \quad !isLargeOrSmallInstance \wedge f(s_{best}) \geq f_{low}) \\ B_i \times \left(e^{-\delta(random \in [min, max])}\right) + \beta & \texttt{otherwise} \end{cases}$$

NLGD has been used for solving timetabling problems such as those shown in Table 2.8.

All studies set the target threshold value, β, to 0 as the lower bound of the objective function. [1] set the remaining parameters based on the size of the instance being solved as summarised in Table 2.9. [101] set $\delta = 5 \times 10^{-7}$ for all instances, whereas the settings for min and max were

Table 2.8: COPs solved using Non-linear Great Deluge (NLGD) move acceptance.

Problem Domain (COP)	**Source(s)**
Exam Timetabling	[100]
University Course Timetabling	[1]
	[101]

set according to the cost of the current solution such that $min = 80,000$ and $max = 90,000$ when $f(s) > 20$, and $min = 20,000$ and $max = 30,000$ when $f(s) \leq 20$. B_{min} and B_{max} were set to 0.85 and 1.5 respectively for small and medium sized instances, whereas large size instances use the respective settings of 1 and 5. [41] set $\delta = 5 \times 10^{-10}$, $B_{min} = 100,000$, and $B_{max} = 300,000$; however, no settings are reported for the parameters min or max. In summary, all studies except that in [41] re-configure NLGD's parameters based on the size of the instance being solved and would suggest that NLGD is especially sensitive to its parameter settings.

Table 2.9: Parameter settings used for NLGD in [1] based on the size of the instance being solved.

	Instance Size		
Parameter	Small	Medium	Large
δ	5×10^{-10}	5×10^{-8}	5×10^{-9}
min	$10,000$	$100,000$	$100,000$
max	$20,000$	$300,000$	$300,000$
B_{min}	2	1	1
B_{max}	5	4	3

2.4.15 Extended Great Deluge (EGD)

Extended Great Deluge [102] is an *adaptive non-stochastic threshold* move acceptance method which is similar to the standard GD, accepting all non-worsening moves and worse moves that are not worse than a threshold value, τ_i, but with the inclusion of a "reheat" mechanism for the threshold value and subsequent decay rate which is activated after a period of non-improvement, as shown in Equation 2.17.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq \max(f(s_i), \tau_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.17}$$

In addition to the standard GD, EGD contains multiple additional parameters which dictates how τ_i is controlled throughout the search.

- A *decay rate* determines what percentage of the computation budget should be used for reducing the initial threshold value to the target threshold value.

- A *wait for non-improvement* parameter decides what percentage of the computational budget should elapse without improving the solution before τ_i should be "reheated".
- A *post-reheat boundary* decides the setting of τ_i after the reheat has occurred.
- A *post-reheat decay rate* decides what percentage of the *remaining* computational budget should elapse to reduce τ_i from its reheated value to the target threshold value. Dashes denote a lack of information in the literature.

EGD has been used for solving timetabling problems such as those shown in Table 2.10.

Table 2.10: COPs solved using Extended Great Deluge (EGD) move acceptance. τ_0 and *decay rate* were set to $f(s_0)$ and 50% of the remaining computational budget respectively for all domains. The settings for *wait for non-improvement* (wni), *post-reheat decay rate* (prdr), and *post-reheat boundary* (prb) are specified below where CB is short for computational budget.

Problem Domain (COP)	**Source(s)**	*wni*	*prdr*	*prb*
Exam Timetabling	[103]	25% total CB	50% remaining CB	$f(s_i)$
	[100]	25% total CB	50% remaining CB	$f(s_i)$
University course timetabling	[104]	-	25% remaining CB	-

τ_0 was set equal to the cost of the initial solution, the decay rate and post-reheat decay rate as 50% of the remaining computational budget, the wait for non-improvement to 25% of the total computational budget, and the post-reheat boundary to be equal to the cost of the current solution at that point of the search in [103] and [100]. τ_0 was also set to be equal to the cost of the initial solution, and the decay rate to 50% of the total computational budget in [104]. The post-reheat decay rate is however set to be equal to 25% of the remaining computational budget. The settings for "wait for non-improvement" was not stated.

2.4.16 Linear Monte Carlo (LMC)

Linear Monte Carlo [57] is a *static stochastic* move acceptance method which accepts non-worsening moves, and randomly accepts worse moves such that a given move is accepted if a random number generated between 0 and 100 does not exceed a value equal to the change in objective value subtracted from a fixed setting M, $M - \Big(f(s^{'}) - f(s)\Big)$.

$$s_{i+1} \leftarrow \begin{cases} s_i' & f(s_i') \leq f(s_i) \vee \text{ random } \in [0,100] < M - (f(s_i^{'}) - f(s_i)) \\ s_i & \text{otherwise} \end{cases} \tag{2.18}$$

LMC was used in [57] as the move acceptance method of a hyper-heuristic for solving the Component placement sequencing for multi head placement machine problem where they fixed the parameter setting of M to be equal to 5 for solving all problem instances, and was chosen based on what obtained the best result during preliminary testing.

2.4.17 Exponential Monte Carlo (EMC)

Exponential Monte Carlo [57] is a *static stochastic* move acceptance method which is the same as the Metropolis algorithm [105]. EMC is different from Simulated Annealing (SA) in that there is no temperature parameter (T) to reduce the probability that worse moves are accepted over time; hence, EMC is a static variant of the better known SA move acceptance method. EMC also has an *adaptive stochastic* definition as proposed in [73] where the inclusion of a re-scaling parameter, shown as an optional parameter k in Equation 2.19, where as $k = 1/T \times \mu_{impr}$ where μ_{impr} is decreased at each improving iteration according to there is no tendency to decrease this over time, hence it is not qualified as an adaptive variant of SA. That is, for the *static* versions of EMC, $k = 1$.

$$s_{i+1} \leftarrow \begin{cases} s_i' & f(s_i') \leq f(s_i) \vee \text{ random } \in [0,1] < e^{-(f(s_i')-f(s_i))k} \\ s_i & \text{otherwise} \end{cases} \tag{2.19}$$

Some studies report to use "Metropolis Acceptance", but it is identical to EMC; hence, they are merged in this survey under EMC as this name was the first occurrence for its usage as a move acceptance method of a hyper-heuristic. EMC has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.11.

Table 2.11: COPs solved using Exponential Monte Carlo (EMC) move acceptance.

Problem Domain (COP)	**Source(s)**	k
Capacitated Vehicle Routing Problem	[55]	1
Component Placement Sequencing For Multi Head Placement Machine	[57]	1
Exam Timetabling	[55]	1
One-dimensional bin packing	[106]	1
Personnel scheduling	[106]	1
Cross Domain Search (CHeSC 2011 domains)	[73]	$\frac{1}{T \times \mu_{impr}}$
	[74]	$\frac{1}{T \times \mu_{impr}}$
	[41]^M	Not stated

EMC was originally used in [57] for solving the component placement sequencing for multi-head placement machine problem where they did not use any scaling factor, i.e. $k = 1$. Further studies were performed [107, 106] where EMC was used as the move acceptance method of a hyper-heuristic for solving two different problems. Both approaches did not utilise a scaling factor. EMC has since been used for solving the cross-domain search problem where [73] proposed the use of a scaling parameter (μ_{impr}) which equates to the moving mean average improvement of previous moves. This scaling parameter is used to re-scale δ across the different domains such that $k = \frac{1}{T \times \mu_{impr}}$. This same approach is used in the subsequent studies using EMC for solving the cross-domain search problem including [74] and [41]. The effects of using different settings for T was performed in [73] where $T \in \{10^{-7}, 10^{-6}, ..., 10^{7}\}$; however, they did not report the best setting as used in their final algorithm. T was set and fixed to 0.5 for all problem domains in [74] and [41]. In conclusion, while the standard static stochastic definition of EMC is used for solving one or two problems, in the literature the adaptive stochastic variant is used by all those approaches tackling the cross-domain search problem.

2.4.18 Exponential Monte Carlo with Counter (EMCQ)

Exponential Monte Carlo with Counter [57] is an *adaptive stochastic* move acceptance method which accepts all non-worsening moves, and worse moves probabilistically such that the larger the objective value change is the less likely it is to be accepted, as shown in Equation 2.20, where δ is the change in solution quality ($f(s^{'}) - f(s)$), t is the elapsed computational time, and Q is the "counter" parameter which increases as the number of consecutive non-improving moves increases.

$$s_{i+1} \leftarrow \begin{cases} s_i' & f(s_i') \leq f(s_i) \vee \text{ random } \in [0,1] < e^{-\frac{\delta t}{Q}} \\ s_i & \text{otherwise} \end{cases} \tag{2.20}$$

EMCQ is an extension of EMC such that the probability that a given worse move is accepted in EMCQ increases as the number of consecutive non-improving moves increases. EMCQ has been used for solving a variety of COPs such as those shown in Table 2.12.

Table 2.12: COPs solved using Exponential Monte Carlo with Counter (EMCQ) move acceptance.

Problem Domain (COP)	**Source(s)**	t
Benchmark function optimisation	[26]	inverse of the *expected range of maximum fitness change*
Component Placement Sequencing For Multi Head Placement Machine	[57]	*computation time*
Exam timetabling	[108]	*elapsed time in minutes*

All papers from the literature that use EMCQ as the move acceptance component of a hyper-heuristic are applied for solving a single problem. Generally, EMCQ can be seen as a parameter-free move acceptance method; however, the elapsed computational time (t) can be specified in different ways. Both [57] and [108] use EMCQ with t representing the elapsed computation time in minutes. The configuration for t used in [26] is equal to the inverse of the *expected range of maximum fitness* *change*. Rather than counting improving moves, Q in [26] is equal to the remaining number of iterations.

2.4.19 Simulated Annealing (SA)

Simulated Annealing [18] is a *dynamic stochastic* move acceptance method which accepts all non-worsening moves, and worse moves based on a probability that decreases both in time as the search progresses, as controlled by the temperature parameter (T), and as the move delta increases ($-\delta$), such that $\delta = f(s_i^{'}) - f(s_i)$, as shown in Equation 2.21. While SA is the oldest example of a move acceptance method (or as referred to in the literature as a metaheuristic), it can be seen as an extension of EMC (see Section 2.4.17) where the scaling factor k as $\frac{1}{T}$ is increased (T is decreased) over the duration of the search.

$$s_{i+1} \leftarrow \begin{cases} s_i' & f\left(s_i'\right) \leq f\left(s_i\right) \vee \text{ random } \in [0,1] < e^{-\delta/T} \\ s_i & \text{otherwise} \end{cases} \tag{2.21}$$

There are multiple cooling schedules for reducing T as used in the literature such as the geometric cooling schedule as used in [63] where T is reduced at each step by multiplying it with a fixed value, α, where $0 < \alpha < 1$, Lundy and Mees's cooling schedule as used in [109] where the temperature is reduced as shown in Equation (2.22) where $0 < \beta < 1$, and a linear descent cooling schedule as used in [110] where the temperature is linearly reduced from an initial temperature at the start of the search, to a final temperature which is to be reached at the end of the search.

$$T_{i+1} = \frac{T_i}{1 + \beta T_i} \tag{2.22}$$

An *adaptive stochastic* definition of SA also exists where a scaling parameterised version of the linear descent cooling schedule is used, and the initial temperature is reduced in time proportional to the cost of the best solution found so far [111].

SA has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.13.

In this review of the parameter settings used for SA, the studies are grouped based on their cooling schedules in order of; geometric cooling, Lundy and Mees's cooling, and linear descent cooling, with cross-domain configurations being discussed towards the end of the review. Those studies which lack any explanation are omitted for conciseness.

Table 2.13: COPs solved using Simulated Annealing (SA) move acceptance.

Problem Domain (COP)	**Source(s)**
Capacitated vehicle routing problem	[55]
Eternity II puzzle	[58]
Exam timetabling	[108]
	[84]
	[55]
High-school Timetabling	[112]
	[60]
Multidimensional 0-1 knapsack problem	[79]
	[80]
Nurse rostering problem	[110]
	[81]
Patient admission scheduling problem	[81]
Personnel routing and rostering	[86]
Personnel scheduling	[63]
Ready-mixed concrete delivery problem	[87]
Sales summit scheduling	[63]
Shelf space allocation	[69]
	[109]
Sports Scheduling	[71]
University Course Timetabling	[111]
	[92]
Cross Domain Search (CHeSC 2011 domains)	[111]
	[112]
	[88]
	[74]
	[41]^{M}

Geometric Cooling The geometric cooling schedule was used in [63, 55, 92] where both [63] and [55] set α equal to 0.85 and the initial temperature is set to half of the objective value of the initial solution. [92] set the initial temperature such that 85% of worse moves are initially accepted and vary the setting for α, comparing its "reliability" to other move acceptance methods.

Lundy and Mees's Cooling Lundy and Mees's Cooling was used in [69, 109] where; [69] set the initial temperature to 30% of the objective value of the initial solution, and the final temperature to 0.1. [109] set the initial temperature such that around 85% of worse moves are accepted, and the final temperature such that only 1% of inferior moves are accepted. β was calculated in both [69, 109] based on an estimated run time of each iteration such that the cooling schedule can be used under a time contract algorithm. A detailed explanation and derivation of the equation can be found in [69].

Linear Descent Cooling The linear cooling schedule was used in [108, 84, 112, 80, 60]. Except where stated, a final temperature setting of 0 was used. The initial temperature was either set as; the objective value of the initial solution [108], or the difference between the objective values of an initially generated solution, and a solution obtained by solving an LP-relaxed version of the targeted problem [80]. [111, 112, 60] sets and updates a scaling factor (ΔF) to *"a factor of the cost of the best solution in hand"*. This scaling parameterised version of the linear descent cooling schedule is shown in Equation 2.23 where ΔF is the scaling factor, i is the current iteration/elapsed time, $T_{elapsed}$ is the elapsed time budget, T_{total} is the total/maximum time budget, and T is the maximum number of iterations/time limit. For this variant of the linear descent cooling schedule, [111] sets ΔF as $0.01 \times f(s_{best_i})$, and [112, 60] sets ΔF depending on whether the solution in hand violates any hard constraints to make the acceptance of infeasible solutions less likely such that [112] uses settings of 0.1 and 0.0001 for ΔF when the solution in hand violates any hard constraint or does not violate any hard constraints respectively, and [60] sets ΔF to 0.01 and 1.00 in the respective scenarios. In contrast, a parameter-free approach was taken in [87, 86, 81, 84] with T representing the remaining computational budget such that T is calculated as $\frac{T_{remaining}}{T_{total}}$ and $T \in [0, 1]$. A similar approach was used in [110] but they parameterised this mechanism such that the initial temperature was set to 1.002 and reduced to 0.002.

$$p = e^{\frac{-\delta}{\Delta F(1-T)}} \texttt{ where } T = \frac{T_{remaining}}{T_{total}} \tag{2.23}$$

Cross-domain Configurations SA based hyper-heuristics have been used for solving the cross-domain search problem, as composed of the CHeSC 2011 competition domains. The configurations of each usage of SA as applied to the cross-domain search problem all use the same scaling parameterised version of the linear descent cooling schedule, but with different settings/mechanisms to determine the scaling factor. [111] and [112] set the scaling factor, ΔF, at each iteration i as $0.01 \times f(s_{best_i})$. [74] use a modified scaling factor (μ_{impr}) which equates to twice the moving mean average improvement of all previous improving moves, and [41] uses the same modified scaling factor but where μ_{impr} equates to the moving mean average improvement of all previous improving moves (not twice).

2.4.20 Simulated Annealing with Reheating (SARH)

Simulated Annealing with Reheating [113] is an *adaptive stochastic* move acceptance method which accepts all non-worsening moves, and worse moves based on a probability that decreases both in time as the search progresses, as controlled by the temperature parameter (T), and as the move delta increases ($-\delta = -(f(s_i^{'}) - f(s_i))$). The acceptance mechanism for SARH is the same as for SA as shown in Equation 2.21 but T can be both decreased or increased depending on the state of the search.

SARH is a variant of SA where the temperature setting is directly increased based on a reheating mechanism. This is in contrast to other adaptive variants of SA such as Very-fast Simulated Re-annealing (VFR) where the algorithmic parameters affecting the temperature setting are controlled to allow the temperature to increase/decrease continually over time - VFR was designed for solving "real-world nonlinear physical problems" [114], but has not been used as the move acceptance method component of a hyper-heuristic before. SARH has been used for solving a variety of COPs such as those shown in Table 2.14.

Table 2.14: COPs solved using Simulated Annealing with Reheating (SARH) move acceptance.

Problem Domain (COP)	**Source(s)**
Exam timetabling	[108]
Nurse rostering	[115]
One-dimensional bin packing	[116] [115]
Shipper rationalisation problem	[117]
Travelling tournament problem	[118]
University course timetabling	[115]

The mechanisms for *when* to re-heat and *how* to re-heat often vary from one study to another.

[118] used SARH under a random choice hyper-heuristic framework for solving the Travelling tournament problem where the geometric cooling schedule was used and the temperature re-heated to twice the value of the temperature setting that was in use at the time of the best solution being found after a consecutive number of moves where the best solution was not improved. [117] use Lundy and Mees's cooling for solving the Shipper rationalisation problem but employ a strategy where the temperature is reduced upon accepting a move, but increase the temperature if the move is rejected. [116] and [108] use the same approach as [115] as discussed below. [115] used SARH under a hyper-heuristic framework for solving three different problems where they take a different approach. They use two temperature control strategies; an annealing phase is used to reduce the temperature as usual according to Lundy and Mees's cooling, and a reheating phase which increases the temperature at each step. This reheating phase is activated after the ratio of accepted moves falls below a pre-defined threshold where the temperature setting is set to that used when the improvement was made, and remains active until a further improvement can be made.

2.4.21 Late Acceptance (LA)

Late Acceptance [28] is an *adaptive non-stochastic basic* move acceptance method which accepts all non-worsening moves, and worse moves whose candidate solution's objective value is not worse than the solution that was current L iterations previous (s_{i-L}), as shown in Equation 2.24.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq max\,(f(s_i), f(s_{i-L})) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.24}$$

LA has been used for solving a variety of COPs, as well as for solving the cross-domain search problem, such as those shown in Table 2.15.

Table 2.15: COPs solved using Late Acceptance (LA) move acceptance.

Problem Domain (COP)	**Source(s)**	L
Exam Timetabling	[28]	$L \in [100, 120000]$
	[119]	500
	[84]	500 or 10000
High-school timetabling	[60]	500
Multidimensional 0-1 knapsack problem	[79]	500
	[80]	500
Personnel routing and rostering	[86]	10
Ready-mixed concrete delivery problem	[87]	500
University course timetabling	[92]	L varied randomly to investigate effects
Cross Domain Search (CHeSC 2011 domains)	[88]	Not stated

LA has been used as the move acceptance component of a hyper-heuristic for solving either a single problem, or for solving the cross-domain search problem. A list length of $L = 500$ was used in [28, 119, 84] for solving the Toronto benchmark instances for Exam timetabling. [28] used an additional approach, again for solving the Toronto benchmark instances, using different settings of L for each instance with the shortest list length being 100, and the longest list length being 120000, such that the settings are empirically determined to produce an approximate run-time of 10 minutes. [84] additionally applied LA for solving the KAHO benchmark instances where L was fixed as 10000. A list length of 500 was also used in [79] and [80] for solving the MKP, and also in [87] for solving the Ready-mixed concrete delivery problem and [60] for solving High-school timetabling problems. A shorter list length of 10 was used in [86] for solving the Personnel rostering and routing problem. [92] varied L in separate trials to study the reliability of LAHC compared to their newly proposed SCHC move acceptance method. The cross-domain search problem was tackled in [88] where they compared their proposed approach to other hyper-heuristics, including one using LA, however the list length setting was not stated. To summarise, a list length of 500 is most often used for LA irrespective of the problem being solved; however, various studies, such as [28] and [86], found it beneficial to use different settings for L.

2.4.22 Late Acceptance with Initial Threshold Accepting (LAIT)

Late Acceptance with Initial Threshold Accepting [74] is an *adaptive non-stochastic threshold* move acceptance method which is identical to Late Acceptance but with the inclusion of an initial period where worse moves whose solution's objective value are not worse than the initial solution (s_0) are accepted for the first T_{wait} period of time of the search as shown in Equation 2.25. LAIT was called as "Accept Late" in [74], however its synonymy with Late Acceptance with the inclusion of an initial threshold accepting strategy, warrants this proposed change in identification - LAIT.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & T_{elapsed} \leq T_{wait} \wedge f(s_i^{'}) \leq f(s_0) \vee f(s_i^{'}) \leq max\left(f(s_i), f(s_{i-L})\right) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.25}$$

LAIT was used in [74] as the move acceptance method of a hyper-heuristic for solving the cross domain search problem (CHeSC 2011 domains) where they set the list length (L) to be equal to $10,000$ for solving the problems from all six domains.

2.4.23 n-Top List (n-TL)

n-Top List [74] is an *adaptive basic* move acceptance method which accepts all moves that are not worse than the n^{th} best solution found so far. Here, n is a fixed parameter which determines the number of best solution values to keep track of, and *best_solutions* is an array/list which holds the n best solution costs found so far throughout the search.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq \max(f(s_i), f(best\_solutions[n])) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.26}$$

[74] used the n-TL move acceptance method under a hyper-heuristic framework for solving the cross-domain search problem (CHeSC 2011 domains) composed of six problem domains; One-dimensional bin packing, Permutation flow shop, Personnel scheduling, Maximum satisfiability, Travelling salesman problem, and Vehicle routing with time windows. In their paper, they used a setting of n equal to 20 for solving all problems and their instances.

2.4.24 n-Best List (n-BL)

n-Best List [74] is an *adaptive basic* move acceptance method which accepts all moves that are not worse than the n^{th} best solution found so far where different solutions of equal quality are not included in the list. That is, n-BL is different to n-TL (see Section 2.4.23) such that solutions of equal quality to the best solution do not displace a record in the current list. Objective values of solutions are only added to the list if they improve over the current best. As with n-TL, n is a fixed parameter which determines the number of best solution values to keep track of, but in this case the best solution values are *unique*. *best_solutions* is an array/list which holds the n unique best solution costs found so far throughout the search.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq \max(f(s_i), f(unique\_valued\_best\_solutions[n])) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.27}$$

[74] used the n-BL move acceptance method under a hyper-heuristic framework for solving the cross-domain search problem (CHeSC 2011 domains) composed of six problem domains; One-dimensional bin packing, Permutation flow shop, Personnel scheduling, Maximum satisfiability, Travelling salesman problem, and Vehicle routing with time windows. In their paper, they used a setting of n equal to 10 for solving all problems and their instances.

2.4.25 Adaptive Iteration Limited List-based Threshold Accepting with a Fixed Limit (AILLA-F)

Adaptive Iteration Limited List-based Threshold Accepting with a Fixed Limit [87] is an *adaptive non-stochastic basic* move acceptance method. AILLA-F accepts all non-worsening moves. A counter for the number of continuously generated worse solutions (*w_iterations*) is used to enable/disable a threshold accepting-based acceptance strategy based on a fixed parameter k which decides the maximum allowed *w_iterations* before the threshold accepting-based acceptance strategy is enabled. Once *w_iterations* falls below this boundary, the threshold accepting-based acceptance strategy is disabled. The threshold accepting-based acceptance strategy maintains a list of n best solutions found where n is a parameter value which is fixed. The list only accepts a new best solution value if the previously found best solution is improved; hence, it is not necessarily the n absolute best solutions found and depends on the order in which they are visited as to whether they are added or not. A control strategy determines which value, N^{th}_{best}, in this list is used for the threshold value. After a number of iterations where the best solution found is not improved (K), which is a parameter that is fixed, N^{th}_{best} is incremented such that the threshold value being used is increased to allow more worse moves to be accepted. When the best solution found is improved, this control strategy is reset such that N^{th}_{best} is 1^{st}_{best} found (i.e. the solution just found).

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee \left(f(s_i^{'}) \leq f\left(s_{N^{th}_{best}}\right) \wedge w\_iterations \geq k \right) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.28}$$

AILLA-F was used in [87] as the move acceptance method of a hyper-heuristic for solving the Ready-mixed concrete delivery problem and they fixed the parameter settings of AILLA-F to be the same for all problem instances where $k = 5, K = 125, n = 10$.

2.4.26 Adaptive Iteration Limited List-based Threshold Accepting (AILLA)

Adaptive Iteration Limited List-based Threshold Accepting [120] is an *adaptive non-stochastic basic* move acceptance method which expands upon AILLA-F from Section 2.4.25 by introducing a re-initialisation strategy, and two control mechanisms for controlling the iteration limit, k, and best solution record length, N, respectively. The re-initialisation strategy is enabled for the first 50% of the computational budget and is activated when $n == N$. The iteration limit, k, is controlled as shown in Equation 2.29 where i is the number of iterations currently performed, and cw is a ratio between the number of elapsed iterations and the current setting for k such that $cw = \lfloor i/k \rfloor$, and

tf is the amount of time remaining scaled between 0 (end) and 1 (start).

$$k = \begin{cases} ((N-1) \times k + i)/N & \text{if } cw == 0 \\ ((N-1) \times k + \sum_{x=0}^{cw} k \times 0.5^x \times tf)/N & \texttt{otherwise} \end{cases} \tag{2.29}$$

The list length N is reduced over time according to Equation 2.30 where N_0 is the initial list length, N_{final} is the target final list length, and tf is the amount of time remaining scaled between 0 (end) and 1 (start).

$$N = N_{final} + (N_0 - N_{final} + 1) \times tf^3 \tag{2.30}$$

The fundamental acceptance strategy of AILLA is shown in Equation 2.31.

$$s_{i+1} \leftarrow \begin{cases} s_{re\text{-}init} & n == N \\ s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee \Big(f(s_i^{'}) \leq f(s_{n^{th}best}) \wedge w\_iterations \geq k \Big) \\ s_i & \texttt{otherwise} \end{cases} \tag{2.31}$$

AILLA move acceptance method was used in [88] under a hyper-heuristic framework for solving the cross-domain search problem (CHeSC domains). They used a single parameter configuration for all domains with $N_0 = 10$ and $N_{final} = 5$, however the rest of the parameter settings were not stated.

2.4.27 An overview of existing methods

There are, to date, 26 different move acceptance methods that have been used under a single-point based hyper-heuristic framework. These move acceptance methods are evenly spread across the different natures of the accept/reject decision from the taxonomy given in Section 2.3. The traditional (older) methods, such as; Simulated Annealing (in 1983), Great Deluge (in 1993), and Record to Record Travel (also in 1993) are classified as either *stochastic* or *non-stochastic threshold*. More recently, *non-stochastic basic* methods have emerged with the earliest of such (excluding those simplistic methods - IE, OI, and AM) being Late Acceptance in 2008, and Step Counting Hill Climbing being introduced as recent as 2016. As can be seen by classifying the existing methods, there are no move acceptance methods which utilise a *dynamic* algorithmic parameter setting mechanism in combination with a *non-stochastic basic* acceptance strategy. It is in fact not possible to design a move acceptance method which would be classified as dynamic non-stochastic

Table 2.16: Overview and classification of move acceptance methods as used in the scientific literature. If a move acceptance method has been used with a different algorithmic parameter setting nature compared to their original descriptions, references of the earliest occurring paper using those alternative mechanisms are given in their respective column(s) and stylised in *italic*. A key can be found in Table 2.17 mapping the abbreviations of each move acceptance method to their given names.

Nature of Acceptance	**Nature of the Parameter Setting**		
	Static	Dynamic	Adaptive
Non-stochastic Basic	AM [37]	-	LA [28]
	OI [37]		AILLA-F [87]
	IE [53]		AILLA [120]
			n-BL [74]
			n-TL [74]
			SCHC [92]
Non-stochastic Threshold	*TA [55]*	TA [91]	RRT [19]
		GD [19]	*GD [95]*
		FD [100]	FD [99]
			BATA [5]
			EGD [102]
			ILTA [93]
			AILTA [85]
Stochastic	EMC [57]	SA [18]	*SA [114]*
	LMC [57]		EMCQ [57]
	NA [89]		*EMC [73]*)
			SARH [113]
			NLGD [1]
			AA [89]
			LAIT [74]

basic since to change one of the static methods into dynamic, one would require the use of the current iteration/time. Without using memory (adaptive), the only possibility is to switch between the existing static non-stochastic basic move acceptance methods (for example AM to IE). However, this would then constitute a multi-stage algorithm with two static non-stochastic basic move acceptance methods - rather than a new move acceptance method itself. The majority of the approaches tend towards those with adaptive algorithmic parameter setting mechanisms, with 17 (+4) out of the 26 move acceptance method either defined originally as adaptive, or (as denoted in parenthesis) having an adaptive variant. This is in contrast to the 6 (+1) with a *static* algorithmic parameter setting mechanism, and the 3 (+1) with a *dynamic* algorithmic parameter setting mechanism. This is unsurprising since it is known that parameter control over parameter tuning is one key prerequisite for improving the performance of evolutionary algorithms [51].

Some of the move acceptance methods from the literature employ strategies to adaptively determine the number of iterations (or amount of time) that a particular setting should be used for, or to activate a different acceptance mechanism. These strategies keep a count of some event

Table 2.17: Key of move acceptance method abbreviations

Abbreviation	Name
AA	Adaptive Acceptance
AILLA	Adaptive Iteration Limited List-based Threshold Accepting
AILLA-F	Adaptive Iteration Limited List-based Threshold Accepting with a Fixed Limit
AILTA	Adaptive Iteration Limited Threshold Accepting
AM	All Moves
BATA	Backtracking Adaptive Threshold Accepting
EGD	Extended Great Deluge
EMC	Exponential Monte Carlo
EMCQ	Exponential Monte Carlo with Counter
FD	Flex Deluge
GD	Great Deluge
IE	Improving or Equals
ILTA	Iteration Limited Threshold Accepting
LA	Late Acceptance
LAIT	Late Acceptance with Initial Threshold Accepting
LMC	Linear Monte Carlo
NA	Naïve Acceptance
NLGD	Non-linear Great Deluge
n-BL	n-Best List
n-TL	n-Top List
OI	Only Improving (sometimes referred to as Improving Only)
RRT	Record-to-record Travel (also stylised RTR)
SA	Simulated Annealing
SARH	Simulated Annealing with Reheating
SCHC	Step Counting Hill Climbing
TA	Threshold Accepting

for which when it exceeds a target value/limit, the settings of the move acceptance method are updated. Examples of such strategies include counting the consecutive number of non-improving moves (AILLA, AILLA-F, AILTA, ILTA), the number of accepted moves (SCHC), and the number of improving moves (SCHC). Some move acceptance methods, such as EMCQ, embed these counters as parameters of the algorithm itself, where the consecutive number of non-improving moves is used to directly influence the probability to accept worse moves.

Selection hyper-heuristics are made up of two key components which are invoked successively; heuristic selection, and move acceptance. Despite the emergence of hyper-heuristics as methods for solving the cross-domain search problem (being reusable and effectively applicable to multiple domains), there are no move acceptance methods in the literature that have been developed specifically for solving this problem - only to complement the heuristic selection process in selection hyper-heuristics. The papers targeting the cross-domain search problem emphasise the design of new heuristic selection techniques over the move acceptance method(s) that they are paired with. This finding is interesting as it is suggested in [26] that the choice of move acceptance method in a selection hyper-heuristic, which commonly takes the form of a local search metaheuristic, has more effect on the performance of hyper-heuristics compared to the embedded heuristic selection method.

2.5 Summary

In this chapter, a taxonomy was provided for classifying move acceptance methods distinguishing them based on two mechanisms; first, the nature of the accept/reject decision, and secondly the nature of the control of the internal algorithmic parameter setting(s). A survey of move acceptance methods was then conducted where a total of 26 distinct methods were identified. A classification of these methods using the proposed taxonomy reveals that the majority of move acceptance methods have been designed with an adaptive nature of the algorithmic parameter settings, and, to date, no move acceptance method exists which has the classification of *dynamic non-stochastic basic*. With an overwhelming number of move acceptance methods, and no extensive analysis of these methods for cross-domain search, a study is needed to assess their performance as a component of a cross-domain search method. In the following chapter, the methodologies and experimental setup is explained for the subsequent studies where one move acceptance method is chosen from each categorisation of the taxonomy and its cross-domain performance is evaluated under the local search metaheuristic framework.

Chapter 3

Methodologies and Experimental Setup

3.1 Introduction

In this chapter, the experimental framework and the methodologies that are used in the subsequent studies to investigate the cross-domain performance of move acceptance methods under a local search metaheuristic framework are defined. The experimental framework is firstly explained in Section 3.2, and the problem domains, problem instances, and their associated move operators are given in Section 3.3. The methods of analysis that are used to evaluate and compare the move acceptance methods are given in Section 3.4, and the parameter tuning approaches and their methodologies are described in Section 3.5. The move acceptance methods that are used in these studies are detailed in Section 3.6, along with their parameter configurations, and this chapter is summarised in Section 3.7.

3.2 Experimental Framework

The studies in this thesis evaluate and compare the cross-domain performance of move acceptance methods. The move acceptance methods are evaluated under a single-point based perturbative local search metaheuristic framework such that as many confounding factors are eliminated as possible from the experimental design that can influence the cross-domain performance of the overall search methods. That is, the move acceptance methods were chosen to be evaluated under a local search metaheuristic framework because:

1. Local search metaheuristics operate at a high-level and hence do not require problem specific knowledge to be able to solve them to a high quality.

2. Local search metaheuristics *can* aspire to the ultimate goal of cross-domain search methods; that is, that they can be used to solve multiple COPs without the need for expert intervention.

3. Local search metaheuristics do not use any form of learning for selecting neighbourhood operators (as present in a hyper-heuristic framework).

4. Local search metaheuristics being single-point based do not require the configuration of the *additional* parameters that are present in population-based metaheuristics, such as population size and replacement strategies.

5. Local search metaheuristics do not need to be concerned by tabu tenures present in Tabu Search based methods, and the embedded move acceptance method will not be affected by differing neighbourhood exploration strategies.

A local search metaheuristic as used in this thesis is of that as defined in Chapter 1, and as outlined by Algorithm 1. The local search metaheuristic framework is made from the HyFlex Framework Java API [20], which was used for the Cross-domain Heuristic Search Challenge (CHeSC) 2011 competition [22]. The HyFlex framework is designed such that the search is performed over the search space of complete and feasible solutions. Where necessary, the low-level heuristic setting *intensity of mutation*, which affects the number of times a heuristic perturbs a solution in a single application, was set to perform a single perturbation of the solution. Some move operators were modified such that the selected move operators are only applied a single time where their implementations did not allow for this using the intensity of mutation setting. That is, for each iteration of the local search metaheuristic, a single perturbation is performed to the solution-in-hand before employing the move acceptance strategy. The problem domains and their instances that are used in this thesis are given in Section 3.3.

An illustration of the framework is shown in Figure 3.1. In keeping with the syntax used in the definition of a local search metaheuristic defined in Chapter 1, a local search metaheuristic using the framework operates as follows. A layer of abstraction called the *domain barrier* separates the domain specific details from the high-level local search metaheuristic, allowing them to be used to solve any COP without the need to modify the algorithm. This domain barrier allows a few abstract details to pass through it, allowing the local search metaheuristic to communicate and invoke methods on the problem domain. A minimal set of perturbative heuristics (hs) are provided in each problem domain which can be *applied* to modify the solution in hand (s), producing a candidate solution ($s^{'}$). An objective function (f) is used to evaluate the quality of a solution from the solution memory, and the solution memory stores three incumbent solutions; a current solution (s), a candidate solution ($s^{'}$), and the best solution (s_{best}). Finally, each problem domain contains a set of problem instances and a single procedure to provide an initial solution (s_0). During the execution of a local search metaheuristic, the following steps are taken with their respective calls across the domain barrier. Firstly, a solution is initialised by signalling to the problem domain to initialise a solution ($s \leftarrow s_0$). A heuristic is then selected uniformly at random (if there are more than one perturbative heuristic), and the heuristic is "applied" to the solution in hand ($s^{'} \leftarrow h(s)$) by signalling to the problem domain to apply the chosen heuristic. The move acceptance method is then queried along with the objective values of the current and candidate solutions. If the move acceptance method informs the local search metaheuristic to accept the candidate solution, then the problem domain is told to accept this move by replacing the current solution with the candidate solution ($s \leftarrow s^{'}$). Depending on whether the computational budget has been exceeded or not, the search will either terminate, returning the best solution found ($returns_{best}$), or the search will continue the iterative process of perturbing the solution-in-hand, and querying the move acceptance method which solution should be carried forward to the next iteration.

The move acceptance method can easily be interchanged with any other move acceptance method that can operate under a local search metaheuristic framework and it is this component that is the focus of this work and that is replaced in the subsequent studies.

The performance and behaviours of the move acceptance methods are compared and contrasted under the local search metaheuristic framework with the aim of observing both the per-domain and cross-domain effectiveness of different move acceptance methods based on the taxonomy given in Section 2.3. To achieve this, a number of methods of analysis are used as discussed in Section 3.4, and different approaches to parameter tuning are utilised as discussed in Section 3.5. This seeks to compare the effectiveness of two "more general" parameter tuning approaches than per-instance tuning for solving various COPs.

The taxonomy proposed in Section 2.3 gives a total of 9 distinct classifications for move acceptance methods. However, to date, those from the literature fall into only 8 of these. To compare the performance of the existing move acceptance methods for cross-domain search, a single move acceptance method is chosen from each of these by choosing one whose move acceptance method

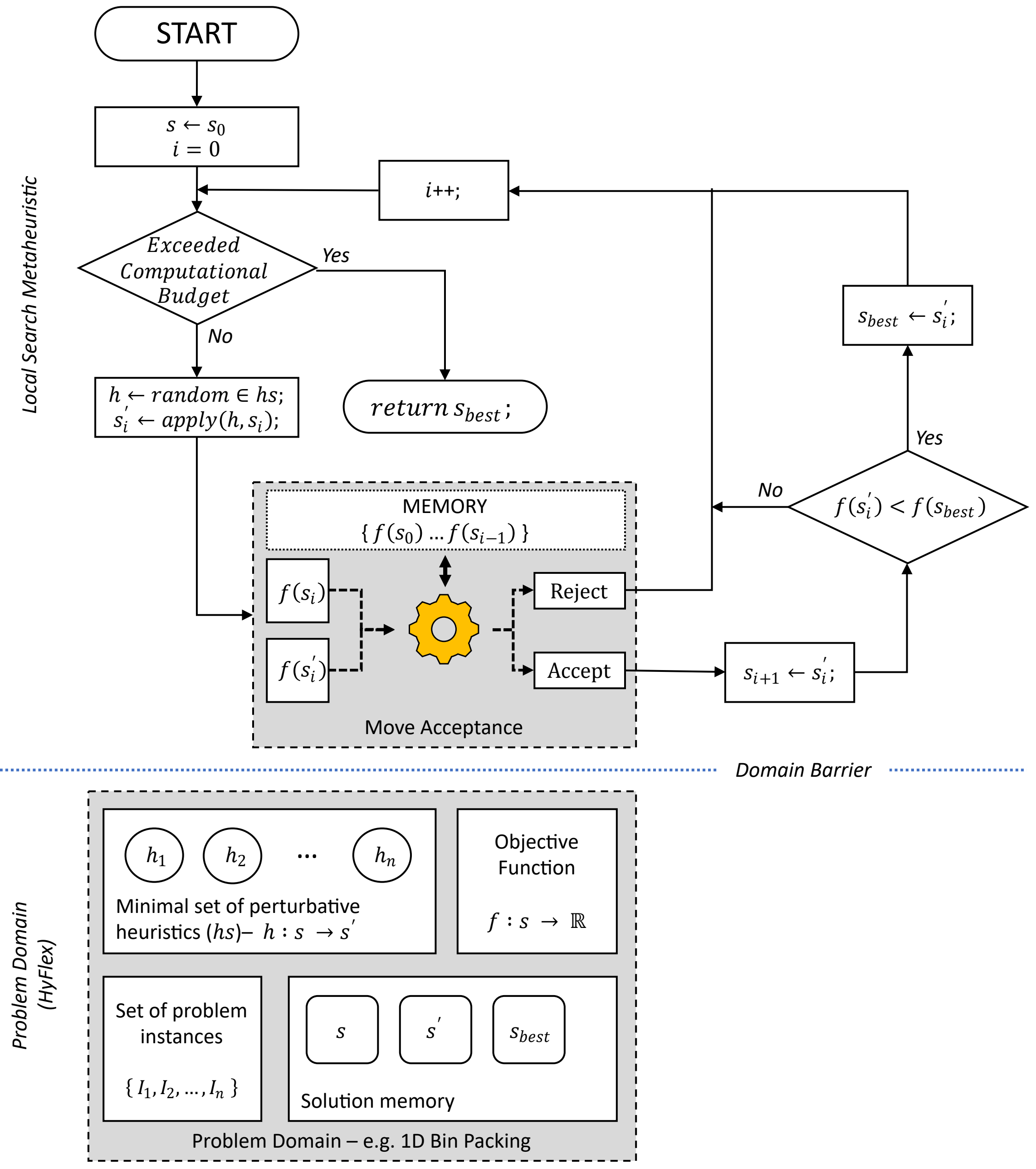

START
$s \leftarrow s_0$
$i = 0$
i++;
Local Search Metaheuristic
Exceeded Computational Budget
Yes
No
$s_{best} \leftarrow s_i'$;
$h \leftarrow random \in hs$;
$s_i' \leftarrow apply(h, s_i)$;
return s_{best};
Yes
No
$f(s_i') < f(s_{best})$
MEMORY
$\{f(s_0) \ldots f(s_{i-1})\}$
$f(s_i)$
$f(s_i')$
Reject
Accept
$s_{i+1} \leftarrow s_i'$;
Move Acceptance
Domain Barrier
Problem Domain (HyFlex)
h_1
h_2
h_n
Minimal set of perturbative heuristics (hs)– $h : s \rightarrow s'$
Objective Function
$f : s \rightarrow \mathbb{R}$
Set of problem instances
$\{I_1, I_2, \ldots, I_n\}$
s
s'
s_{best}
Solution memory
Problem Domain – e.g. 1D Bin Packing

Figure 3.1: Illustration of the local search metaheuristic framework with the move acceptance component being highlighted. The move acceptance method itself, emphasised in yellow, is the interchangeable method which is investigated in these studies.

has previously been used for solving the cross-domain search problem where appropriate, otherwise a well known method is chosen. The chosen move acceptance methods are detailed in Section 3.6.

3.3 Problem Domains

The cross-domain search problem entails having to solve multiple characteristically different COPs. For this reason, all nine available HyFlex compatible problem domains, which to date are still of scientific interest, are used to compare the cross-domain performance of the move acceptance methods in this work. Six of these domains were used within the Cross-domain Heuristic Search Challenge (CHeSC) 2011[1] and are part of the original HyFlex framework [20]. These are, along with references to some recent studies on them, the One-Dimensional Bin Packing Problem (BP) [121, 122], Permutation Flow Shop Problem (FS) [123, 124], Personnel Scheduling Problem (PS) [125, 126, 127], Maximum Satisfiability Problem (SAT) [128, 129], Travelling Salesman Problem (TSP) [130, 131, 132], and the Vehicle Routing Problem (VRP) [133, 134]. The remaining three domains were recently introduced [135] to the HyFlex Framework as a HyFlex Extension (HyFlext). These are the 0-1 Knapsack Problem (KP) [136, 137, 138], Max Cut Problem (MAC) [139, 140, 141], and Quadratic Assignment Problem (QAP) [142, 143]. Previous studies on cross-domain search targeted minimisation COPs where the objective function is quick to evaluate, in other words, they are not inverse optimisation problems. Therefore the problems considered below are minimisation problems with non-inverse objective functions. The problem domains, objective functions (denoted as $f(x)$, where x is a given solution to the problem), initialisation methods, move operators, and problem instances that are used in the subsequent studies are detailed below.

3.3.1 One-Dimensional Bin Packing (BP)

The One-Dimensional Bin Packing (BP) problem is part of the HyFlex framework and was developed by [144].

Objective Function

The objective of the BP problem is to minimise the number of bins required to pack all pieces from a set of pieces where each piece j has a weight w_j, and each bin has a capacity C, subject to the capacity of each bin not being exceeded, and each piece is contained in exactly one bin. The objective function used is as follows:

[1]The Cross-domain Heuristic Search Challenge 2011 `http://www.asap.cs.nott.ac.uk/external/chesc2011`

$$f(x) = 1 - \left(\frac{\sum_{i=1}^{n} (fullness_i / C)^2}{n} \right)$$

such that

$$fullness_i = \sum_{j=1}^{n_{bi}} w_{b_{i,j}}$$

where

n denotes the number of bins,

C denotes the bin capacity,

$b_{i,j}$ denotes the j^{th} piece in bin i,

n_{bi} denotes the number of pieces in bin b_i, and

$w_{b_{i,j}}$ is the weight of the j^{th} piece in bin i.

Initialisation

BP uses as its initialisation procedure a randomised first fit algorithm [145] whereby the pieces are ordered randomly before a first-fit algorithm iteratively packs each piece into the next available bin that it fits into.

Operators

The set of move operators used as the local search metaheuristic framework are restricted to swap, split, and destroy.

- Swap - selects two pieces at random and, space permitting, places them into each others bin. If one of the pieces that was selected cannot fit into the other bin, then a new bin is created where it is then inserted.
- Split - selects a bin with a higher than average number of pieces, creates a new and empty bin, and then inserts half of the number of pieces into the new bin.
- Destroy - randomly selects the lowest or highest filled bin and uses the best-fit algorithm to repack the pieces from the selected bin into the remaining bins. If a piece cannot fit into any of the remaining bins, then a new bin is created containing the pieces that could not be re-packed.

Instances

A total of five instances are chosen from the available set of instances as those used in the CHeSC competition. The details of these instances can be found in Table 3.1

Table 3.1: One-Dimensional Bin Packing Instances.

Instance ID#	Instance Name	Ref.
1	falkenauer/u1000-01	[146]
9	testdual7/binpack0	[146]
11	testdual10/binpack0	[146]
7	triples2004/instance1	[147]
10	50-90/instance1	[147]

3.3.2 Permutation Flow Shop (FS)

The Permutation Flow Shop (FS) problem is part of the HyFlex framework and developed by [148].

Objective Function

The objective of the FS problem is to find a permutation, π, of a given number of jobs, n, which are to be processed in-order on a set of machines, m, in a pre-defined machine ordering whereby the makespan, C_{max}, is minimised. Each job is processed by each machine, M, in the order $[M_0, M_1, ..., M_{m-1}]$. Once a job is available to be processed, any given machine must start processing that job and should not remain idle unless there are no jobs for the machine to process. The objective function used is as follows:

$$f(x) = C_{max}$$

where

$$p_{i,j} \text{ is the processing time of the job } i \text{ on machine } j,$$

$$\pi \text{ is the permutation of } n \text{ jobs,}$$

$$\pi(q) \text{ is the index of the job in the } q^{th} \text{ position in the permutation } \pi,$$

$$C_{\pi(q),j} \text{ is the completion time of the job } \pi(q) \text{ on machine } j,$$

$$C_{\pi(0),0} = p_{\pi(0),0},$$

$$C_{\pi(q),0} = C_{\pi(q-1),0} + p_{\pi(q),0} \text{ for } q > 0,$$

$$C_{\pi(0),j} = C_{\pi(0),j-1} + p_{\pi(0),j} \text{ for } j > 0,$$

$$C_{\pi(q),j} = p_{\pi(q),j} + max\{C_{\pi(q-1),j}, C_{\pi(q),j-1}\} \text{ for } q > 0 \wedge j > 0,$$

$$C_{max} = C_{\pi(n-1),m-1}.$$

Initialisation

FS uses as its initialisation procedure the randomised NEH algorithm [149] whereby a list containing a random permutation of jobs is created. Each job is taken in order from this list and added to a new schedule in the position which generates a partial solution with the smallest makespan.

Operators

The solution of an instance from the FS domain is represented using a permutation representation. Therefore, the only move operator required for the local search metaheuristic framework is a swap operator.

- Swap - randomly selects two (unique) jobs from the permutation and swaps their positions.

Instances

A total of five instances are chosen from the available set of instances as those used in the CHeSC competition. These instances were taken from [150], and the HyFlex instance ID#'s and instance names can be found in Table 3.2:

3.3.3 Personnel Scheduling (PS)

The Personnel Scheduling (PS) problem is part of the HyFlex framework and developed by [151].

Table 3.2: Permutation Flow Shop Instances.

Instance ID#	Instance Name
1	20x5/2
3	100x20/4
8	500x20/2
10	200x20/1
11	500x20/3

Objective Function

The objective of the PS problem is to assign each employee from a group of employees a set of shifts over a specified planning horizon subject to several coverage and employee working objectives and constraints. The constraints can be classified as hard and soft constraints, where the hard constraints have to be satisfied, while soft constraints represent preferences, i.e. the employee working objectives. The set of problems covered by Personnel Scheduling are diverse, each with different objectives and constraints. The implementation of the domain in this framework was therefore implemented as a general case domain with all objectives and constraints built-in where all constraints are converted to highly weighted objectives, and the weights of objectives adjusted depending on the particular instance being solved. In general, the overall objective function can be defined as shown below summing up all constraint violations to be minimised, where the weights are changed based on the particular characteristics of the instance.

$$f(x) = \sum_{i=1}^{G} w_i g_i(x)$$

where

G is the number of constraints,

g_i measures the degree of the violation(s) of the i^{th} constraint, and

w_i is the weight for g_i

These constraints are described below, and the formulations are presented in detail in [151]:

- The minimum and maximum number of hours worked.
- The minimum and maximum number of days on or off.
- The minimum and maximum number of consecutive working days.
- The minimum and maximum number of consecutive days off.

- The minimum and maximum number of consecutive working weekends.
- The minimum and maximum number of consecutive weekends off.
- The minimum and maximum number of shift of a certain type; e.g. night shifts.
- The minimum and maximum number of consecutive shifts of a certain type.
- Shift rotations; e.g. an early shift should be avoided after a night shift.
- Satisfying employee requests such as specific days or shifts on or off.

Initialisation

A solution in the PS domain is initialised using the 'new' local search operator [151]. The solution initially consists of no assignments of employee shifts to days. Shifts are removed and assigned to employees for each day in the planning period of increasing length, where length is the number of consecutive days worked on the same shift type, iteratively. It does this by trying increasing shift lengths for each employee, day, and then shift type, by changing the assignment of a block of consecutive shifts of size equal to the current shift length being considered, irrespective of shift type, for the current employee being tried, to the current shift type being tried. If this results in an improvement in the roster, then the change in assignment is kept, otherwise it is reverted. This process is repeated until all swaps (changes in shift type assignments) have been tried.

Operators

The PS problem uses a specialised solution representation. In order to satisfy the local search meta-heuristic framework, the required operators are *new swap*, *vertical swap*, and *random unassign*[2].

- New Swap - A local search operator which introduces or removes a shift-block to the roster.
- Vertical Swap - Swaps a shift pattern of length l where $l \leftarrow random \in \{x \mid x \in \mathbb{Z}, 1 \leq x \leq 5\}$ between two employees.
- Random Unassign - Unassigns a random shift from the roster but while maintaining solution feasibility.

[2]Thanks to Tim Curtois for his expertise in the implementation of this domain and for providing us with the information relating to the required move operators.

Instances

A total of five instances are chosen from the available set of instances as those used in the CHeSC competition. The details of these instances can be found in Table 3.3

Table 3.3: Personnel Scheduling Instances.

Instance ID#	Instance Name	Ref.
5	Ikegami-3Shift-DATA1.2	[152]
8	ERRVH-B	[153]
9	MER-A	[153]
10	BCV-A.12.1	[153]
11	ORTEC01	[153]

3.3.4 Maximum Satisfiability (SAT)

The Maximum Satisfiability (SAT) problem is part of the HyFlex framework and developed by [154].

Objective Function

The objective of the SAT problem is to assign a truth value to each variable, v, within a set of variables which, given a set of clauses, c, minimises the total number of unsatisfied clauses as represented in conjunctive (clause) normal form. That is, each clause contains at least one variable, and each variable is contained in at least one clause. The objective function is as follows:

$$f(x) = \sum_{i=1}^{N} c_i$$

where

$$N \text{ is the total number of clauses, and } c_i = \begin{cases} 0 & \text{if the } i^{th} \text{ clause evaluates to true} \\ 1 & \text{otherwise} \end{cases}$$

Initialisation

SAT, being represented as a binary string, generates a random binary string as its initialisation procedure such that each variable in the SAT problem has a random truth value.

Operators

The solution of an instance from the SAT domain is represented as a binary string. Therefore, the only move operator required for the local search metaheuristic framework is a flip operator.

- Flip - randomly selects a single variable and negates its current truth value.

Instances

A total of five instances are chosen from the available set of instances as those used in the CHeSC competition. The details of these instances can be found in Table 3.4

Table 3.4: Maximum Satisfiability Instances.

Instance ID#	Instance Name	Ref.
3	parity-games/instance-n3-i3-pp	[155]
4	parity-games/instance-n3-i3-pp-ci-ce	[155]
5	parity-games/instance-n3-i4-pp-ci-ce	[155]
10	jarvisalo/eq.atree.braun.8.unsat	[156]
11	highgirth/3sat/hg-3sat-v300-c1200-4	[157]

3.3.5 Euclidean Travelling Salesman Problem (TSP)

The Euclidean Travelling Salesman Problem (TSP) is part of the HyFlex framework. There is no official documentation for the TSP domain implementation, but the reader is referred to [158] for general information on the TSP problem itself.

Objective Function

The objective of the TSP problem is to find a closed route between a set of cities such that each city is visited exactly once, and the route returns to the starting city whereby the total distance is minimised. More specifically, for the current formulation, the objective is the find a permutation, π, of cities, c, whereby the total distance between all adjacent cities, including the first and final cities, $d(i,j)$, are minimised. The distance function $d(i,j)$ calculates the non-rounded Euclidean distance between the cities c_i and c_j, where x_{coord} and y_{coord} are the coordinates representing the location of a given city. The objective function used is as follows:

$$f(x) = d(\pi(n-1), \pi(0)) + \sum_{i=0}^{N-2} d(\pi(i), \pi(i+1))$$

where

$$N \text{ is the total number of cities,}$$

$$\pi(l) \text{ is the index of the city in the permutation index } l \text{ for } 0 \leq l < N,$$

$$d(i,j) = \sqrt{(c_i.x_{coord} - c_j.x_{coord})^2 + (c_i.y_{coord} - c_j.y_{coord})^2} \text{ for } 0 \leq i,j < N.$$

Initialisation

TSP, being represented using a permutation representation, generates a random permutation of cities as its initialisation procedure such that the order in which the cities are visited is randomised.

Operators

Similar to the FS domain, the only move operator required for the local search metaheuristic framework of a solution with permutation representation is a swap operator.

- Swap - randomly selects two (unique) cities from the permutation and swaps their positions.

Instances

A total of five instances are chosen from the available set of instances as those used in the CHeSC competition. These instances were taken from [159], and the HyFlex instance ID#'s and instance names can be found in Table 3.5:

Table 3.5: Travelling Salesman Problem Instances.

Instance ID#	Instance Name
0	pr299
2	rat575
6	d1291
7	u2152
8	usa13509

3.3.6 Vehicle Routing with Time Windows (VRPTW)

The Vehicle Routing with Time Windows (VRPTW) problem is part of the HyFlex framework and was developed by [160, 161].

Objective Function

The objective of the VRPTW problem is to find one or more routes, each serviced by a delivery vehicle, which between them are able to visit a set of customers exactly once subject to time-window and vehicle capacity constraints, whilst minimising the number of vehicles required, and the total distance travelled. Each route starts and ends at the same location which is defined as the depot location and visits at least one customer location. Each customer must be serviced within their time window, and the capacity of each vehicle should not be exceeded. Note that these constraints are handled by the low-level heuristics to maintain solution feasibility. The objective function used is as follows:

$$f(x) = 1000v + \sum_{i=0}^{v-1} \sum_{j=0}^{|route_i|-1} d(route_i(j), route_i((j+1) \mod |route_i|)$$

where

$$v \text{ is the number of vehicles (routes),}$$
$$route_i \text{ is the route for the } i^{th} \text{ vehicle,}$$
$$|route_i| \text{ is the length of the route } route_i,$$
$$route_i(j) \text{ is the } j^{th} \text{ location in route } route_i,$$
$$d(m, n) \text{ is the Euclidean distance between locations } m \text{ and } n$$

Initialisation

The initialisation method is a randomised constructive approach which uses a metric which contains a stochastic element which considers the distance and time of each customer to the most recently inserted customer to sequentially select a customer and insert them into the partial solution until all customers have been feasibly scheduled. If it is not possible to insert any more customers into the current route (corresponding to a single vehicle), then a new route is generated.

Operators

There are two parts to the representation of a solution for this domain. The first is the routes themselves, and within each route, a permutation of customers which are unique across all routes. That is, no customer should belong to more than one route. Two operators are required to be able to reach all possible solutions within the local search metaheuristic framework; 2-opt mutation to change the order of a single route, and shift mutation to swap customers between routes.

- 2-opt mutation - swaps two adjacent customers within a route subject to time window constraints.
- Shift mutation - Removes a single customer, c, from a randomly chosen route, r, and tries to place them into another route, r' where $r' \neq r$, subject to time window and vehicle capacity constraints. The customer is tried in all positions in the new route in a random order until the first feasible solution is found. If the move cannot result in a feasible solution, then a new route is created with the customer as a single delivery.

Instances

A total of five instances are chosen from the available set of instances as those used in the CHeSC competition. These instances were taken from [162], and the HyFlex instance ID#'s and instance names can be found in Table 3.6:

Table 3.6: Vehicle Routing with Time Windows Instances.

Instance ID#	Instance Name
1	Soloman/RC/RC207
2	Soloman/RC/RC103
5	Homberger/RC/RC2-10-1
6	Homberger/R/R1-10-1
9	Homberger/C/RC1-10-8

3.3.7 0-1 Knapsack Problem (KP)

The 0-1 Knapsack (KP) problem was introduced as part of the HyFlex extension set (HyFlext) and developed by [163].

Objective Function

The objective of the KP problem is to find a subset, $s \in S$, of a set of items, where each item has an associated weight, w, and profit, p, which maximises the total profit while adhering to the capacity constraint, V, of the knapsack. The following objective function used formulating KP as a minimisation problem.

$$f(x) = \frac{1}{\sum_{i \in s} p(i) + \alpha}$$

where

$$s \subseteq S,$$

$$V \text{ is the volume of the knapsack}, V > 0,$$

$$\sum_{i \in s} w(i) < V,$$

$$w(i) \text{ is the weight of item } i \texttt{ for } w > 0, w \in \mathbb{R},$$

$$p(i) \text{ is the profit of item } i \texttt{ for } p > 0, p \in \mathbb{R},$$

$$\alpha \text{ is a fixed small value of } 1 \times 10^{-64} \text{ to prevent cases of division by 0.}$$

Initialisation

A greedy procedure is used which iteratively packs the next most valuable item into the knapsack until no more items can be inserted.

Operators

In the 0-1 knapsack problem, a set of items are given. Each item is either assigned to be in, or to not be in, the knapsack. Two operators are required to be able to reach all possible solutions within the local search metaheuristic framework; pack random, and remove random. These operators are each used with a 50% probability by the local search metaheuristic framework such that an item is either added to or removed from the knapsack with equal chance.

- Pack random - chooses a random item from the set of items which are both not already in the knapsack and can fit into the knapsack, and packs it into the knapsack.
- Remove random - removes a random item from the knapsack.

Instances

A total of five instances are chosen randomly from the available set of instances such that there are a mixture of small, medium, and large sized instances. These instances were taken from [164], and the HyFlex instance ID#'s and instance names can be found in Table 3.7:

Table 3.7: 0-1 Knapsack Problem Instances.

Instance ID#	Instance Name
0	1K-15-12
1	2K-1-13
3	2K-5-17
5	5K-1-24
8	5K-5-28

3.3.8 Max Cut Problem (MAC)

The Max Cut (MAC) problem was introduced as part of the HyFlex extension set (HyFlext) and was developed by [165].

Objective Function

MAC is a set-partitioning problem where the objective is to find a partition (p_1 and p_2) of a set of vertices, V, from a weighted connected graph, G, with edges E, each of which has a weight, w, that maximises the sum of the weights of the edges that cross the partition. The following minimising objective function is used in this study.

$$f(x) = \frac{1}{\sum_{e \in E_{p_1,p_2}} w(e) + \alpha}$$

where

$$p_1 \subseteq V, p_2 \subseteq V, p_1 \cup p_2 = V, \text{ and } p_1 \cap p_2 = \emptyset,$$

E_{p_1,p_2} is the set of edges that connect vertices which are not in the same partition,

$w(e)$ is the weight of edge e,

α is a fixed small value of 1×10^{-64} to prevent cases of division by 0.

Initialisation

A greedy randomised constructive procedure is used in which each vertex is selected at random and iteratively inserted into the partition that maximises the cost of the cut between the two disjoint sub-graphs of the partial solution.

Operators

In the max cut problem, there are a set of vertices which must be partitioned into two disjoint subsets. A single move operator, swap random, is required to reach all possible solutions within the local search metaheuristic framework.

- Swap random - moves a randomly selected vertex from one partition to the other.

Instances

A total of five instances are chosen randomly from the available set of instances such that there are a mixture of small, medium, and large sized instances. The details of these instances can be found in Table 3.8

Table 3.8: Max Cut Problem Instances.

Instance ID#	Instance Name	Ref.
0	g3-8	[166]
2	g14	[167]
5	g22	[167]
7	g55	[167]
9	pm3-15-50	[166]

3.3.9 Quadratic Assignment Problem (QAP)

The Quadratic Assignment (QAP) problem was introduced as part of the HyFlex extension set (HyFlext) and developed by [168].

Objective Function

QAP is an assignment problem where the objective is to find an assignment of n facilities, F, to a set of n locations, L where each facility is assigned to a unique location, such that the sum of the products of the distance, d, and flows (weights), w, between all facilities are minimised.

$$f(x) = \sum_{\forall a,b \in F} (w(a,b) \times d(l_a, l_b))$$

where

$$w(a,b) \text{ calculate the weight (flow) between two facilities } a \text{ and } b,$$

$$l_i \text{ is the location for the facility } i \in F,$$

$$d(l_a, l_b) \text{ returns the distance between two locations, } l_a \text{ and } l_b$$

Initialisation

In the Quadratic Assignment problem, a set of n facilities are to be assigned to a single and distinct location from a set of m locations such that the sum of the distances multiplied by the flows between each location and facility is minimised. The initialisation of a solution involves uniformly randomly assigning one of the facilities to a location such that each facility has a single associated location, and each location has either one or no associated facilities.

Operators

A single move operator, swap random, is required to reach all possible solutions within the local search metaheuristic framework.

- Swap random - chooses two facilities at random (a, b) with locations (l_a, l_b) and exchanges their assigned locations such that a's location becomes l_b and b's location becomes l_a.

Instances

A total of five instances are chosen randomly from the available set of instances such that there are a mixture of small, medium, and large sized instances. These instances were taken from [169], and the HyFlex instance ID#'s and instance names can be found in Table 3.9:

Table 3.9: Quadratic Assignment Problem Instances.

Instance ID#	Instance Name
0	sko100a
6	tai150b
7	tai256c
8	tho150
9	wil100

3.4 Methods of Analysis

To evaluate the cross-domain performance of each move acceptance method throughout the work in this thesis, a separate local search metaheuristic embedding each of the move acceptance methods (and their configurations) is used to solve a total of 45 problem instances, 5 instances from each of the 9 problem domains as covered above in Section 3.3. The local search metaheuristics, embedding each move acceptance method, are evaluated 31 times on each problem instance where the termination criteria for each run (evaluation) is equal to 10 nominal minutes. This is equivalent to 415 seconds on our machine, as determined by the official benchmark tool[3], using an Intel Core i7-3820 processor at 3.60GHz with 16GB of memory running Windows 10 and Java 1.8.0_40-b26. The number of trials and computational budget are set in line with those used in the CHeSC 2011 competition. This allows us to not only compare the different move acceptance methods under the local search metaheuristic framework to each other, but also to the results of the existing state-of-the-art methods.

There are two general strategies (approaches to parameter tuning) that are used to compare the cross-domain performance of each move acceptance method; the first strategy considers how they perform when their parameters have been re-tuned for each problem being solved, and the second strategy considers how they perform when their parameters have been tuned a single time and the parameter configuration used for solving problems from all domains. These strategies are called by their tuning approaches of *per-domain tuned*, and *cross-domain tuned.* These are more-general than more involved tuning methods known as instance-specific, or per-instance, tuning methods where the search method is re-tuned for each and every problem instance [27], where either an expert is involved in the re-tuning process, or time is taken to for each problem instance to perform automated tuning on them. Without having to perform computationally expensive parameter tuning experiments for all problem instances, and while *reducing* the expert intervention efforts, the *per-domain* tuning approach aims to show the "best-case" cross-domain performance of each move acceptance method. The ultimate goal in cross-domain search research however is to eliminate the need for expert intervention entirely. Cross-domain tuning is therefore used as an approach which does not require an expert to re-tune the parameters of the move acceptance methods as it uses a single parameter configuration determined from a subset of the training instances. The cross-domain tuning approach is used to show the cross-domain performance of each move acceptance method when used as a cross-domain search method in its expected use case - without expert

[3]HyFlex benchmarking tool available online: http://www.asap.cs.nott.ac.uk/external/chesc2011/benchmarking.html

intervention. These approaches to parameter tuning (referred to as tuning strategies) are described in further detail in Section 3.5.

The work in this thesis aims to observe the cross-domain performance of the local search methods embedding the characteristically different (based on our taxonomy) move acceptance methods. One of the problems faced when comparing general-purpose search methods over multiple domains (and even problem instances) is that the different problem domains and instances have a different range of objective values. When comparing their performance using their "raw" results, the problem instance(s) with the highest magnitude of objective values dominate those with values of lower magnitudes. In order to be able to compare the performances of the move acceptance methods across different domains, the results are normalised following the scheme used in [72], and as shown in Equation (3.1) where $f(s)$ is the result being normalised, and $f(s_{best})$ ($f(s_{worst})$) is the best (worst) solution obtained by all algorithms over the same problem instance. The normalised results are thereby linearly scaled between 0 (best result) and 1 (worst result) for each problem instance.

$$f_{norm}(s) = \frac{f(s) - f(s_{best})}{f(s_{worst}) - f(s_{best})} \tag{3.1}$$

The *cross-domain* performance of each move acceptance method is calculated as the sum of the normalised results ($f_{norm}(s)$) from all 45 problem instances and their trials, equating to 1395 results per move acceptance method, and this metric is referred to as its μ_{norm} score [135]. The benefit of using μ_{norm} scores over ranking methods, such as the Formula 1 ranking mechanism used in the CHeSC 2011 competition is that the normalised $f_{norm}(s)$ result proportionally represents the performance of each move acceptance method compared to each other move acceptance method. For example, consider three hypothetical algorithms A_1, A_2, and A_3 with normalised results of 0.10, 0.15, and 0.85 respectively. Ranking would only show that A_1 outperforms A_2 and A_3, and A_2 outperforms A_3. The normalisation metric additionally shows that A_2 does not perform much worse than A_1, but much better than A_3. The μ_{norm} metric is therefore chosen as a more meaningful and reflective cross-domain performance score for comparing the cross-domain performance of the move acceptance methods to each other. In addition, a *per-domain* performance score is used to compare the performance of each move acceptance method across multiple problem instances over the same problem domain. This score is calculated as the sum of normalised $f_{norm}(s)$ scores across all 5 instances of the respective problem and we call this as its $\nu_{norm}(d)$ score where d is the problem domain; that is, 155 results per move acceptance method.

Lilliefors test was performed on the computational results to test for normality and showed that they do not always come from a normal distribution. The move acceptance methods in this thesis are therefore compared using non-parametric statistical tests. All statistical tests are performed using a confidence interval of 95%. To compare the statistical significance between the performances of two algorithms, the Wilcoxon signed rank test is used on the $f_{norm}(s)$ results that are re-calculated over both algorithms. To compare the statistical significance between the performance of more than two algorithms, Kruskal-Wallis one-way ANOVA test was initially used before learning of a more suitable statistical test, the Friedman test, which was used in [170] for comparing evolutionary algorithms across a set of problems, and additionally considers repeated measures (in this case, multiple trials for the same problem instance). The post-hoc tests of both Kruskal-Wallis one-way ANOVA and Friedman tests were conducted using Bonferroni-Dunn's correction procedure, as suggested by [170]. Both multiple comparison tests are able to be applied on the actual results, rather than requiring normalised results, and hence these values are used directly and without adjustment for such testing.

3.5 Parameter Tuning

3.5.1 Approaches to Parameter Tuning

Since one of the objectives of cross-domain research is to reduce/eliminate the need for the expert role to design and re-configure solution methods for solving different COPs, two 'more-general' parameter tuning configurations are evaluated in this work. That is to say, per-instance parameter tuning is not considered in this work due to the necessity to have to re-tune each search method for each and every problem and their instances. This approach to parameter tuning is not a common practice in the development of search methods even considering a single domain. The two parameter tuning approaches that are used are *per-domain tuning* and *cross-domain tuning*. In *per-domain tuning*, the parameters of the move acceptance methods are re-tuned for solving each of the nine problem domains, and each instance within that problem is solved using the same parameter configuration. That is to say, that the same parameter configuration is used to solve all instances of the same problem. Parameter tuning is performed on a subset of problem instances, 1 small and 1 large sized instance per domain, as defined in Table 3.10. The parameter configuration that is obtained is used by the move acceptance method for solving all five instances of the respective problem domain. In *cross-domain tuning*, the parameters of the move acceptance methods are tuned a single time on a set of eight problem instances, 1 small and 1 large from four domains.

They are the same instances as used for per-domain tuning, but from only the problem domains; BP, FS, PS, and SAT. These domains were chosen for cross-domain tuning as they were the tuning domains used in the HyFlex competition. That is, the remaining five domains remain "unseen" - maintaining a key ideology of cross-domain search methods in that they can be used to solve new and unseen problems to a high quality. This parameter configuration is used by the respective move acceptance method for solving all instances from all problem domains from the cross-domain search benchmark.

Table 3.10: Training problem instance ID#'s used for parameter tuning experiments.

Domain	Training Instance ID#'s
Bin Packing	1, 11
Flow Shop	1, 11
Personnel Scheduling	5, 9
Maximum Satisfiability	5, 11
Travelling Salesman Problem	2, 8
Vehicle Routing Problem	1, 6
0-1 Knapsack Problem	0, 8
Max Cut Problem	0, 9
Quadratic Assignment Problem	0, 9

3.5.2 Tuning Methodologies

There are multiple requirements for parameter tuning of the move acceptance methods that are used in these studies. Some move acceptance methods contain no parameters (i.e. IE and NA) and are hence used as-is. Others were designed such that they do not have parameters that are meant to be tuned (i.e. AILLA and AILTA). That is, their parameters are designed to be internal to the algorithm and were previously configured for the CHeSC 2011 competition which has the same computational budget as that used in these studies; hence, their default and intended parameter configurations were reused.

The remaining move acceptance methods either contain 1 (TA, GD), 2 (SA), 3 (SARH), or 4 (HAMSTA) parameters.

A discrete parameter space was considered for the parameter tuning experiments due to the complexity and computational expense of a continuous parameter space. This is especially exacerbated by the 10 nominal minute evaluation time required for each trial. For move acceptance methods that have 1 or 2 parameters, a full factorial design of experiments (DOE) is used. The Taguchi design of experiments [171] is used for tuning the parameters of move acceptance methods that contain more than 2 parameters. The Taguchi DOE has been used previously for parameter optimisation such as in [172] where the parameters of PSO were tuned for benchmark function optimisation, and in [49] where the parameters of a steady-state memetic algorithm were tuned for cross-domain search.

Each move acceptance method is tuned by performing 31 trials on each training instance for a total of 10 nominal minutes per trial (in line with the CHeSC 2011 competition) for each parameter configuration. The results obtained from all parameter configurations are then normalised using the $f_{norm}(s)$ metric and analysed based on the respective parameter tuning experiment. For the full factorial DOE, the best parameter configuration is chosen as the one which resulted in the lowest μ_{norm} score across the training instances. For per-domain tuning, this is the sum of the $f_{norm}(s)$ results across the two training instances for the respective domain as shown in Table 3.10. For cross-domain tuning, this is the sum of the $f_{norm}(s)$ results across the eight BP, FS, PS, and SAT training instances (1 small and 1 large instance from each domain).

For the Taguchi DOE, an additional step has to be taken to evaluate the estimated performance of all the possible parameter configurations across all the parameters being tuned. The Taguchi DOE is based on orthogonal arrays (OA) which reduces the computational budget of the experiments by considering a fractional factorial DOE. A number of factors (parameters) and levels (possible configurations for each parameter) first have to be identified and the respective OA design is used. For example, SARH uses an L25 OA design which is able to tune 3 factors (parameters), with 5 levels (settings) per factor. Since the Taguchi DOE is a fractional design, only a fraction of possible parameter configurations need to be evaluated. In the case of SARH, the computational budget for per-domain tuning for all 9 domains is reduced from 125 configurations (5^3) equivalent to a computational expense of 11,625 nominal hours (≈484 days) to a more feasible budget of 2,325 hours (≈97 days) using the L25 OA. In order to tune the move acceptance methods for the best cross-(per-)domain performance, $f_{norm}(s)$ scores are calculated for each of the tested parameter configurations considering all configurations of the same move acceptance method for each instance. For each configuration of each parameter, the sum of $f_{norm}(s)$ scores are used to judge their performance. The configuration used for move acceptance methods with three parameters are the parameter settings that yielded the lowest sum of $f_{norm}(s)$ scores across the two (eight) training instances for per-domain (cross-domain) tuning. An example plot showing the performance of each factor and level of SARH for per-domain parameter tuning on the Personnel Scheduling problem is shown in Figure 3.2. There are three factors in this design, the initial temperature calculated as the percentage of accepted worsening moves, χ_0, the final temperature also calculated as the percentage of accepted worsening moves, χ_n, and the factor of the total computational budget, *wait_time*, which should elapse after the best solution is found (or since the previous re-heat) before the temperature is re-heated. Each parameter contains five levels. The lower the mean $f_{norm}(s)$ score, the better that level performed on average over all tuning experiments. The best parameter configuration from these results are hence chosen as that with the lowest mean $f_{norm}(s)$ score from each parameter (factor).

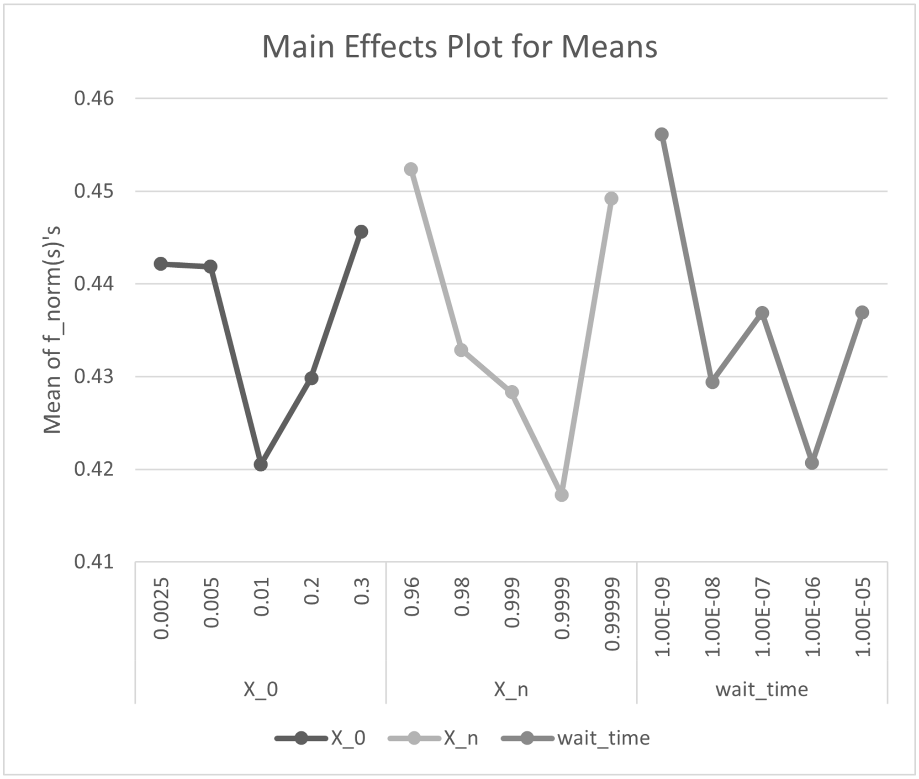


Figure 3.2: Per-domain (PS) performance of the different parameter configurations of SARH as determined by the Taguchi DOE for the per-domain tuning approach. The best parameter configuration chosen in this case is: $\chi_0 = 0.01 (1\%)$, $\chi_n = 0.9999 (99.99\%)$, and $wait\_time = 1.0 \times 10^{-6}$.

The parameter configurations that were obtained for each of the move acceptance methods are detailed alongside their move acceptance methods in Section 3.6.

3.6 Move Acceptance Methods

A total of eight move acceptance methods are chosen for comparison based on the different natures of the accept/reject decision, and natures of the algorithmic parameter settings, covering each part of the taxonomy in Section 2.1, and as classified in Table 2.16. A single move acceptance method was chosen from each classification within the taxonomy based on which was used in the literature for tackling the cross-domain search problem or based on their popularity. These move acceptance methods are detailed in alphabetical order along with their per-domain and cross-domain parameter configurations as obtained by the parameter tuning experiments detailed in the above Section 3.5.

3.6.1 Adaptive Iteration Limited List-based Threshold Accepting (AILLA)

AILLA [120] is an adaptive non-stochastic basic move acceptance method. AILLA maintains a list of length L recording the objective values of the L best solutions found. Whenever the current best solution is improved during the search, the worst objective value is removed from the list and replaced with that of the new best solution. AILLA also incorporates a restart mechanism to reinitialise the current solution depending on several factors of the current search state and is explained in detail in [120]. AILLA accepts a candidate solution as the current solution in the next iteration, as shown in Equation (3.2), if its objective value is not worse than the current solution, or if it is not worse than the ith objective value in the list, as detailed in [120], since the last re-initialisation, and a number of non-improving iterations have passed which is dynamically reduced from 10 to 5 as the search progresses. Note that by $s_{i^{th}best}$, we are referring to the parameter i in the paper for AILLA and not the current iteration.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee \Big(f(s_i^{'}) \leq f(s_{i^{th}best}) \wedge w\_iterations \geq k\Big) \\ s_{re\text{-}init} & \text{(see [120])} \\ s_i & \texttt{otherwise} \end{cases} \tag{3.2}$$

Parameter Configuration

The implementation and parameter settings for AILLA were taken from [120]. All parameters and adaptation methods used are set to their default settings since AILLA was designed as a parameter-less move acceptance method for solving the cross-domain search problem and uses the same termination criterion as in this work. Moreover, it was used as the move acceptance component of the winning hyper-heuristic from the CHeSC 2011 competition, which to date remains one of the best cross-domain search procedures.

3.6.2 Adaptive Iteration Limited Threshold Accepting (AILTA)

AILTA [85] is an adaptive non-stochastic threshold move acceptance method. It is similar to AILLA but with one key difference. AILTA calculates a threshold value based on an ordered list of factors of the best solution, rather than reusing the objective values of the L best solutions found. AILTA accepts a candidate solution as the current solution in the next iteration if the objective value of the candidate solution is not worse than the current solution, or if a certain number of consecutive rejected moves have occurred and the objective value of the candidate solution is less than an acceptance threshold calculated as a factor of the cost of the best solution found so far as shown in Equation (3.3).

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee \\ & w\_iterations \geq k \wedge f(s_i^{'}) \leq f(s_{best}) + |f(s_{best}) \times \epsilon| \\ s_i & \texttt{otherwise} \end{cases} \tag{3.3}$$

Parameter Configuration

The original equation for calculating the threshold from [85] works only for problems whose objective functions return non-negative values. The equation was therefore modified from $f(s_{best}) + (1 \times \epsilon)$ to $f(s_{best}) + |f(s_{best}) \times \epsilon|$ to allow it to work over all objective value ranges.

The ϵ parameter controls how much worse a solution is allowed to be before it is rejected, and k determines how many continuous rejected moves should elapse before switching from the improving or equal acceptance strategy to the threshold accepting strategy. In this work, the setting for k was the same as used in [85] which was 100. The ϵ parameter is incremented according to a number of consecutive non-improving moves exceeding a parameter max_iter and reset to its initial value upon accepting an improving move. In that paper, this was set to 5000 however it was used exclusively for the homecare scheduling problem taking approximately 62500 iterations. Due to the variable iteration count encountered in cross-domain search, if the total number of iterations does not exceed 62500, then max_iter is set proportional to an estimated number of iterations for each problem instance based on pre-experimental analysis as shown in Equation (3.4) rounded to the nearest integer value. Upon exceeding this limit, ϵ is incremented from its initial setting of 0.003 by 0.001 up to a maximum setting of 0.010. As these parameters are internal to the move acceptance method, these settings are used for both the per-domain and cross-domain algorithm configurations.

$$max\_iter = min\left(\frac{5000 \times total\_iterations}{62500}, 5000\right) \tag{3.4}$$

3.6.3 Great Deluge (GD)

GD [19] is a dynamic non-stochastic threshold move acceptance method. A dynamic time-based version of GD from [97] is used in this work assuming a fixed computational budget of T_{total} to run the algorithm. GD accepts a candidate solution as the current solution if the objective function value of the candidate solution ($f(s_i')$) is not worse than the current solution ($f(s_i)$), or if it is not worse than a threshold value (τ_i). This threshold value is linearly decreased considering the elapsed time ($T_{elapsed}$) between the objective function value of the initial solution ($f(s_0)$) and some target value ($qualityLB$), as shown in Equation (3.5) where the acceptance threshold is calculated as in [97] shown in Equation (3.6).

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq \max(f(s_i), \tau_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{3.5}$$

$$\tau_i = qualityLB + (f(s_0) + qualityLB) \times \left(1 - \frac{T_{elapsed}}{T_{total}}\right) \tag{3.6}$$

Parameter Configurations

GD contains a single parameter ($qualityLB$). This parameter should be set as the expected (or known) final objective value such as the lower bound of the problem being solved [97]. In this study, the focus is on investigating the cross-domain performance of different move acceptance methods with the aim of producing a move acceptance method that can be used for solving new and unknown problems with little expert knowledge. In those cases, such lower bounds or optimal values may not be known; therefore, $qualityLB$ is tuned for each problem domain as described in Section 3.5. The range of values used for tuning were between 1×10^{-10} and 1×10^{13}, including 0, with intervals of an increasing order of magnitude.

Per-domain Parameter Configuration The parameter settings for $qualityLB$ determined from the per-domain tuning approach are given in Table 3.11.

Table 3.11: Values of $qualityLB$ that were used for each problem domain using the Great Deluge move acceptance method.

Parameter	BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP
$qualityLB$	1×10^{-9}	1×10^{10}	100	1	1×10^{5}	1×10^{5}	0	1×10^{-7}	100

Cross-domain Parameter Configuration Given the cross-domain parameter tuning approach, a single parameter setting should be determined for $qualityLB$. In this work, the move acceptance methods are evaluated on problems that are minimisation, and that have objective functions that return non-negative values. The value for $qualityLB$ is a target value which should be aimed for when solving a problem, and for the case of solving minimisation-based COPs, this target value is set to 0 as an ideal solution penalty.

3.6.4 Improving or Equal (IE)

IE is a static non-stochastic basic move acceptance method. IE accepts a candidate solution as the current solution in the next stage if and only if the objective function value of the candidate solution is not worse than the current solution as shown in Equation (3.7).

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{3.7}$$

Parameter Configuration

IE does not contain any parameters that can be configured; hence, IE is used in its original form for both per-domain and cross-domain configurations.

3.6.5 Naïve Acceptance (NA)

NA [89] is a static stochastic move acceptance method. NA accepts a candidate solution as the current solution in the next iteration if the objective function value of the candidate solution is strictly better than the current solution, else it accepts the candidate solution with a fixed probability as shown in Equation (3.8).

The only parameter in NA is the probability to accept equal or worse quality solutions. Previous studies mostly fixed this value to be 0.5 as is the case in [89] which evaluated NA for cross-domain search, hence we use this value in this work for both the per-domain and cross-domain tuning variants.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) < f(s_i) \vee random \in [0,1] < 0.5 \\ s_{i+1} & \texttt{otherwise} \end{cases} \tag{3.8}$$

Parameter Configuration

NA contains a single parameter which determines the (fixed) probability that a non-improving solution is accepted. This probability however is mostly used as 0.5 in the literature as the naïve approach accepting half worse moves and rejecting the other half. Moreover, [89] used NA for the cross-domain search with this same configuration. For both per-domain and cross-domain configurations, this probability is therefore fixed at 0.5.

3.6.6 Simulated Annealing (SA)

SA [18] is a dynamic stochastic move acceptance method. SA accepts a candidate solution if its quality is better than or equal to the cost of the current solution, or if a random number in the range $[0, 1]$ is less than some probability P determined by the metropolis criterion [105] as shown in Equation (3.9). The metropolis criterion has a single parameter. This is the system temperature, T, and is decreased in time by an accompanying annealing schedule. Annealing schedules from the literature include linear cooling, geometric cooling and Lundy and Mees cooling [173]. The cooling schedule and procedure for setting the initial and final temperatures of SA are those from [174] and uses a time-based formulation of the geometric cooling schedule, and temperatures are calculated using a sample from the search space of worse moves such that a given percentage of worse moves are accepted. This percentage is denoted as χ.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) \vee random \in [0,1] < e^{-\delta/T} \\ s_i & \texttt{otherwise} \end{cases} \tag{3.9}$$

Parameter Configurations

A full factorial DOE was used for tuning the parameters χ_0 and χ_n of SA. Usually, a high temperature setting is required at the start of the search, and is normally decreased to a small value close to 0. For both per-domain and cross-domain tuning, the parameter settings that were evaluated initially were $\chi_0 \in \{90\%, 92\%, 94\%, 96\%, 98\%\}$ and $\chi_n \in \{0.0625\%, 0.125\%, 0.25\%, 0.5\%, 1\%\}$. In the case that one of the extreme values were determined by the tuning methodology, the values of χ_0 and/or χ_n were expanded in the appropriate direction until a setting was found that was not on a limit. For example if chi_n was tuned to 0.0625%, then the tuning experiments were performed again with $\chi_n \in \{0\%, 0.015625\%, 0.03125\%, 0.046875\%, 0.0625\%\}$. In the case that χ_n was tuned to 0% or the next largest value, a new tuning range was established to test between these values.

Per-domain Parameter Configuration The parameter settings used for χ_0 and χ_n for each domain, as determined from the tuning methodology, are shown in Table 3.12.

Cross-domain Parameter Configuration Performing the full factorial DOE parameter tuning experiments on the cross-domain training instances, the cross-domain parameter configuration that

Table 3.12: Values of χ_0 and χ_n that were used for each problem domain for Simulated Annealing.

Parameter	BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP
χ_0	92%	92%	99.99999999%	92%	40%	60%	80%	92%	70%
χ_n	0%	0%	70%	$\frac{1}{64}$%	0%	$\frac{1}{64}$%	$\frac{1}{32}$%	0%	$\frac{1}{64}$%

was obtained was with $\chi_0 = 98\%$ and $\chi_n = 1.5625 \times 10^{-13}\%$.

3.6.7 Simulated Annealing with Reheating (SARH)

SARH [113] uses an adaptive stochastic move acceptance method. SARH extends upon the traditional definition of Simulated Annealing by periodically increasing the temperature of the system during the search to either prevent it from becoming stuck or to escape from local optima. There are multiple approaches in the literature for deciding the value of the reheated temperature setting, how the cooling schedule operates after reheating, and when a reheat should occur. Such ways in which the reheated temperature is set includes a function of the temperature when the maximum specific heat occurs [175], the initial temperature setting [176], and some factor of the temperature setting when the best solution was found, for example $1.0 \times T_{best}$ or $2.0 \times T_{best}$ in [113] and [118] respectively. After performing a reheat, the cooling schedule can either be left to operate as is, decreasing the temperature over time [175], or disabled such that the temperature is fixed at the reheat temperature for the remainder of the search [113]. Note that reheating is not the same as re-annealing [114] where the parameters affecting the temperature setting are controlled to gradually decrease and increase the temperature over time depending on the search features. Strategies for deciding when to perform the reheat include doing so when the maximum specific heat occurs in the system [175], by keeping track of when the last improvement was made, reheating the temperature after a predefined number of continuous non-improving moves have occurred [113], and after a predefined number of moves since the best solution was last improved [118].

In this work, we use the earliest example of such mechanism for deciding when to perform reheating as it can be seen in other adaptive approaches within the literature to signal an adaptation event such as in AILLA and AILTA. For the reheat temperature, we chose to use the approach used in [118] where the temperature is set to twice that when the best solution was found as preliminary testing showed this to be more effective. After performing a reheat, the cooling schedule is allowed to continue for the remainder of the search. SARH as used in this work contains three parameters; the initial temperature, T_0, the final temperature, T_{final}, and the time since the last improving move was made before applying reheating, *wait_time*. The temperature settings are derived in the same way as for the dynamic version of Simulated Annealing (Section 3.6.6 by using the tuned percentage of worse moves to accept parameters χ_0 and χ_n. The value α, which is used my the cooling schedule to control the rate of cooling is calculated initially and upon each reheat as T_{final}/T_0 and $T_{final}/(2 \times T_{best})$ respectively. The *wait_duration* is calculated as a product of the computational time budget and the *wait_time* parameter where *wait_time* $\in [0, 1]$, exclusive and inclusive.

The mechanism for move acceptance is the same as that used by SA in Equation (3.9). The adaptation procedure for dealing with reheating of the system temperature is shown in Algorithm 2 and an updated version of the cooling schedule to deal with the reheating capability and time-based termination criterion is shown in Equation (3.10).

Algorithm 2: *process*$_2$() for Simulated Annealing with Reheating.

```
if move was accepted then
    if $f(s').isBetterThan(f(s))$ then
        $time\_reheat \leftarrow wait\_duration + time\_elapsed$;
    end
    if $f(s').isBetterThan(f(s_{best}))$ then
        $t_{best} \leftarrow t_i$;
    end
else if $time\_elapsed > time\_reheat$ then
    $t_0 \leftarrow t_{best} \times 2$;
    $time\_reheat \leftarrow wait\_duration + time\_elapsed$;
    $time\_previous\_reheat \leftarrow time\_elapsed$;
    $\alpha \leftarrow t_{final}/t_0$;
return $s_{best}$;
```

$$T = T_0 \times \alpha^{\frac{time\_elapsed - time\_previous\_reheat}{time\_total}} \tag{3.10}$$

Parameter Configurations

The Taguchi DOE method used for tuning the parameters of SARH used an L25 orthogonal array which is a 3 factor with 5 level design. The parameter setting ranges used initially for χ_0, χ_n, and *wait* in the Taguchi DOE were as follows: $\chi_0 \in \{90\%, 92\%, 94\%, 96\%, 98\%\}$, $\chi_n \in \{1\%, 0.5\%, 0.25\%, 0.125\%, \%0.0625\}$, and *wait* $\in \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

Per-domain Parameter Configuration The parameter settings obtained for each domain using the per-domain parameter tuning approach are given in Table 3.13.

Cross-domain Parameter Configuration The cross-domain parameter configuration that was obtained by the Taguchi DOE using the same setup as the per-domain tuning methodology and

Table 3.13: Values of χ_0, χ_n, *wait_time* that were used for each problem domain in the Simulated Annealing with Reheating move acceptance method.

Parameter	BP	FS	PS	SAT	TSP	VRPTW	KP	MAC	QAP
χ_0	70%	96%	99.99%	50%	80%	60%	80%	94%	60%
χ_n	0.015625%	0%	1%	0.0015625%	0%	0.015625%	0.25%	0.000015625%	0.015625%
wait	0.80	0.60	0.000001	0.90	0.40	1.0	0.9999	0.9999	0.20

was $\chi_0 = 96\%$, $\chi_n = 0.0625\%$, and $wait = 0.50$.

3.6.8 Threshold Accepting (TA)

TA [91] is a static non-stochastic threshold move acceptance method. TA accepts a candidate solution as the current solution in the next iteration if and only if the objective function value of the candidate solution is not worse than an acceptance threshold calculated as the sum of the objective function value of the *current* solution and a threshold parameter, T, as shown in Equation (3.11).

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq f(s_i) + T \\ s_i & \texttt{otherwise} \end{cases} \tag{3.11}$$

TA has a single parameter, T, which defines how much worse than the current solution a candidate solution is allowed to be before being rejected. A static definition of TA from [55] is used in this work, which they incorrectly identify as record-to-record travel. This variant was used for solving multiple problems including examination timetabling and the capacitated vehicle routing problem. As a general-purpose search method however, the setting for T must be set appropriately for the problem being solved. The objective function values of different problem domains, and even problem instances, return a different range of values meaning that a single setting would not be effective. The threshold parameter is therefore calculated as a factor (k) of the cost of the initial solution as shown in Equation (3.12). This approach for calculating a threshold setting has been seen in related non-stochastic threshold move acceptance methods such as AILTA, albeit that their threshold values are calculated adaptively.

$$T = k \times f(s_0) \tag{3.12}$$

Parameter Configurations

TA as used in this work contains a single parameter (k) that is used to determine the threshold value as a factor of the cost of the initial solution. The initial testing range for k was between 0.05% and 20% such that $k \in \{0.0075, 0.0050, 0.0025, 0.0010, 0.0005, 1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}, 1 \times$

$10^{-8}\}$. In the event that k was tuned to the lowest ($1E-8$) or highest (0.0075) values, extended ranges of k were considered in the ranges $[0.0, 0.0005]$ and $[0.0075, 1 \times 10^{10}]$ respectively.

Per-domain Parameter Configuration The parameter settings for k for the per-domain tuning approach are given in Table 3.14. For solving Bin Packing problems, a setting of $k = 0$ was far superior to very small values of k. This is a particularly interesting observation, since it shows that on certain problems, certain move acceptance methods can be changed into other methods based on extreme values of their parameter. In this case, TA is converted into IE with the parameter setting of $k = 0$. A similar phenomenon is also observed for solving Knapsack problems where the value of k was tuned to 1×10^8 which is so high that no moves are ever rejected; TA in this case turns into the accept all moves method.

Table 3.14: Values of k that were used to calculate the threshold parameter T for each problem domain for the Threshold Accepting move acceptance method.

Parameter	BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP
k	0.0	1E-4	0.8	1E-3	1E-3	1.5E-4	1E8	5E-4	1.5E-4

Cross-domain Parameter Configuration Using the settings of $k \in \{0.0005, 0.0010, 0.0025, 0.0050, 0.0075, 0.0100, 0.0500, 0.1000, 0.1500, 0.2000\}$, the cross-domain parameter configuration obtained for TA was with $k = 0.0010$.

3.7 Summary

This chapter served to explain the local search metaheuristic framework that is used in the subsequent studies in this thesis to evaluate the move acceptance methods. The nine problem domains, forty-five problem instances, and the perturbative move operators that are used to evaluate the move acceptance methods under the local search metaheuristic framework were then also detailed. The methods of analysis that are used to compare the cross-domain performance of the move acceptance methods are also discussed and how the normalisation of results can be used to perform parameter tuning despite the issue faced with tuning across the characteristically different problem domains and instances. Finally, the eight existing move acceptance methods are explained along with their per-domain and cross-domain tuned parameter configuration(s). Some observations are also discussed alongside the reporting of the parameter configurations concerning when they have been tuned to extreme values. The following chapter evaluates the cross-domain performance of the move acceptance methods discussed in this chapter, each based on a different "nature of the accept/reject decision” and “nature of how the algorithmic parameters are set” pair as defined in the taxonomy under the proposed local search metaheuristic framework.

Chapter 4

On the Cross-domain Performance of Move Acceptance Methods

4.1 Introduction

In this chapter, the performance and behaviours of eight existing move acceptance methods under a local search metaheuristic framework are compared and contrasted. The aim of this comparison is to observe the per-domain and cross-domain effectiveness of the move acceptance methods based on their classification in the taxonomy proposed in Chapter 2.3. The experimental setup and the explanations and configurations of the move acceptance methods can be found in the previous chapter. The move acceptance methods in Section 4.2 use the parameter configurations obtained from the *per-domain parameter tuning approach* such that their *potential* as a general-purpose search method, as if a parameter configuration oracle existed, can be observed. Section 4.2 discusses the results from evaluating each move acceptance method across all instances from the same problem domain (per-domain performance) in Section 4.2.1 to Section 4.2.9. A summary of the move acceptance methods per-domain performance is discussed in Section 4.2.10, and their cross-domain performance is discussed in Section 4.2.11.

4.2 An analysis of the performance of move acceptance methods for cross-domain search

This section covers the results and analysis of the local search metaheuristics embedding different move acceptance methods for cross-domain search. Sections 4.2.1 to 4.2.9 discuss the results for the move acceptance methods solving each problem domain individually (per-domain comparison) using a higher-level per-domain parameter configuration. These discussions include respective parameter settings for each move acceptance method, and analysis of the progress plots (Figure 4.3) and move acceptance statistics (Table 4.3 and Table 4.4) for the best general move acceptance method for each of the nine problem domains. The progress plots are presented to gain insight into the behaviours of each move acceptance method which allows them to solve each problem well. The per-domain and cross-domain scores are given in Table 4.1 and boxplots of the normalised results from all 5 instances per domain are given in Figure 4.1 and Figure 4.2. Results from the Kruskal-Wallis one-way ANOVA tests performed for each problem domain can be found in Table 4.2. The non-normalised results for each problem instance from the experimentation can be found online at: `http://dx.doi.org/10.13140/RG.2.2.15671.14245`. A summary of the results is given in Section 4.2.10, and the cross-domain performance of the move acceptance methods are then discussed in Section 4.2.11.

Table 4.1: Per-domain scores for each move acceptance method over the 9 problem domains, calculated as explained in Section 3.2, with the best general-purpose method for each problem domain stylised bold. The final column shows the cross-domain score for each move acceptance method, calculated as the sum of per-domain scores.

	BP	FS	PS	SAT	TSP	VRPTW	KP	MAC	QAP	Cross-domain
IE	**0.09**	**0.52**	3.63	0.54	**1.98**	0.50	5.00	0.55	0.50	13.31
AILLA	0.85	0.56	2.54	0.08	2.15	3.43	**1.98**	4.23	0.20	16.03
TA	0.39	0.64	0.77	0.54	2.69	**0.17**	4.92	2.29	**0.16**	12.56
GD	3.35	2.92	1.56	3.61	2.83	3.33	4.91	3.12	3.60	29.23
AILTA	0.77	2.52	3.62	0.54	2.73	0.24	5.00	0.69	0.29	16.40
NA	3.26	3.88	**0.58**	3.55	3.13	4.5	4.24	4.24	4.06	31.44
SA	0.36	0.68	0.67	0.09	2.49	0.92	2.54	0.56	0.34	**8.62**
SARH	3.02	0.68	0.70	**0.05**	2.45	0.92	2.49	**0.24**	0.34	10.88

4.2.1 Bin Packing

The best move acceptance method for solving Bin Packing (BP) problems according to the per-domain scores was IE. A boxplot comparison of the performance of the move acceptance methods for solving BP instances can be found in Figure 4.1. The results of performing an ANOVA test on

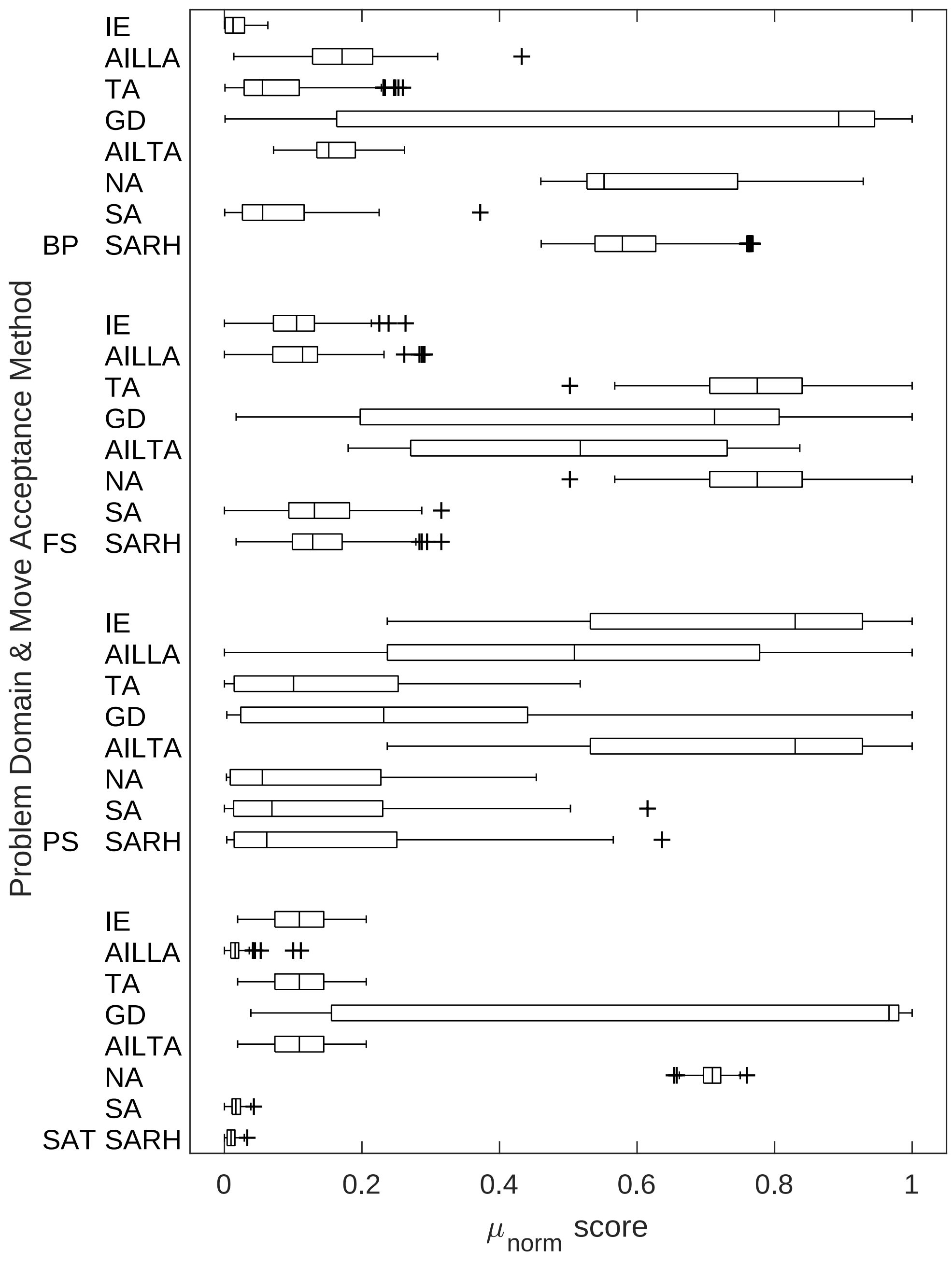


Figure 4.1: $f_{norm}(s)$ values obtained by each move acceptance method using per-domain parameter configurations over all 31 trials for all 5 instances for each of the Bin Packing, Flow Shop, Personnel Scheduling, and Maximum Satisfiability problem domains. '+' marks symbolise statistical outliers according to either $f_{norm}(s) > q_3 + 1.5 \times (q_3 - q_1)$ or $f_{norm}(s) < q_1 - 1.5 \times (q_3 - q_1)$ where q_1 and q_3 are the 25th and 75th sample data percentiles.

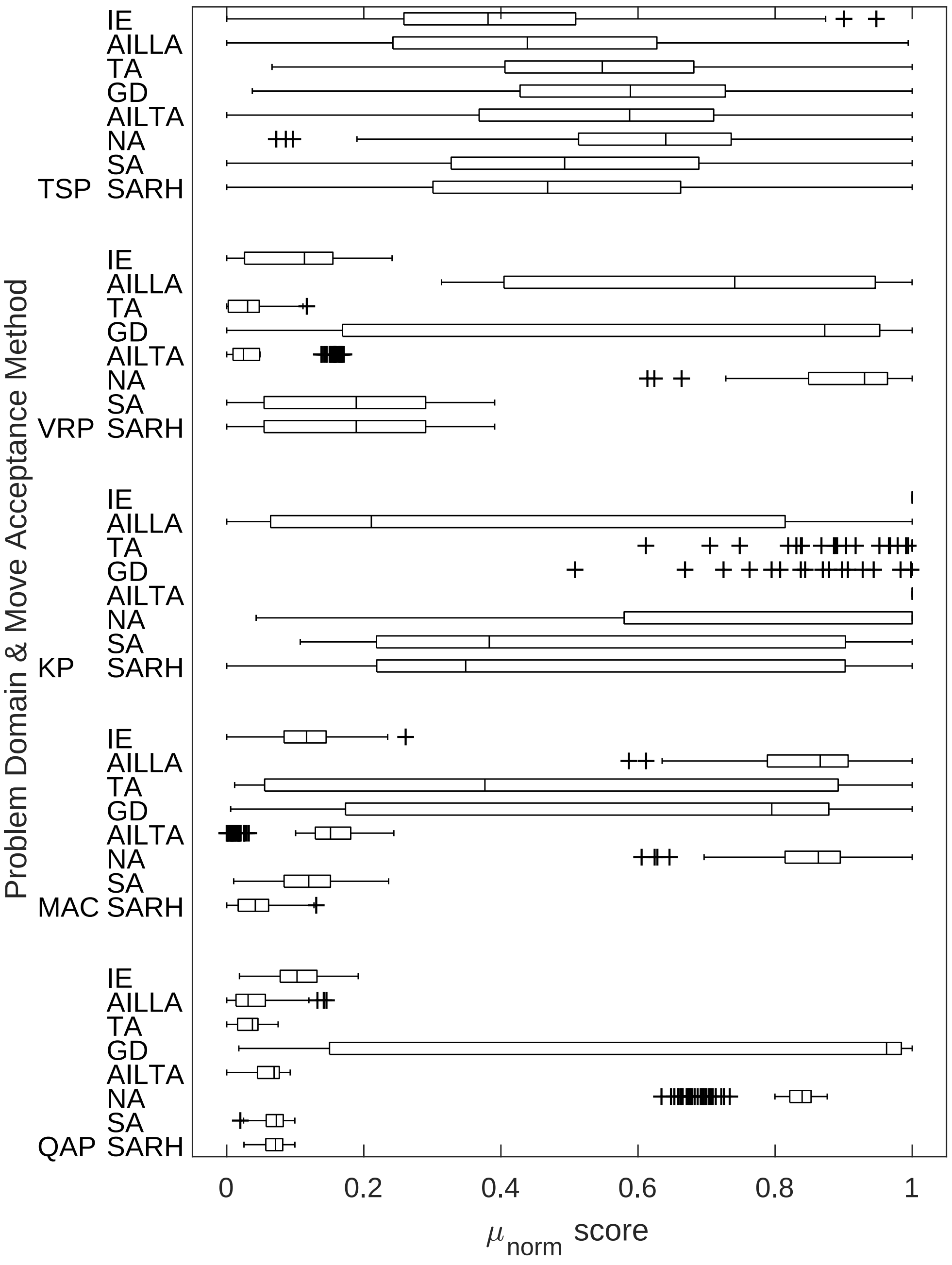


Figure 4.2: $f_{norm}(s)$ values obtained by each move acceptance method using per-domain parameter configurations over all 31 trials for all 5 instances for each of the Travelling Salesman, Vehicle Routing, Knapsack, Max Cut, and Quadratic Assignment problem domains. '+' marks symbolise statistical outliers according to either $f_{norm}(s) > q_3 + 1.5 \times (q_3 - q_1)$ or $f_{norm}(s) < q_1 - 1.5 \times (q_3 - q_1)$ where q_1 and q_3 are the 25th and 75th sample data percentiles.

Table 4.2: Kruskal-Wallis One-way ANOVA comparing the performance of the move acceptance methods for each *problem domain* with n_0 that all results are from the same distribution at CI = 95%. The values are the mean ranks (lower is better) of the aforementioned test with the best move acceptance method, and those which do not statistically significantly differ from the best, for each domain being stylised bold.

Problem	IE	AILLA	TA	GD	AILTA	NA	SA	SARH	$\chi^2(7)$	p
BP	**354.3**	593.5	**431.3**	822.3	550.4	903.1	**417.4**	891.8	420.4	1.03E-86
FS	**529.2**	**536.8**	**550.3**	718.8	729.4	786.5	**555.8**	**557.1**	93.7	2.10E-17
PS	796.5	715.2	**530.0**	**606.5**	796.1	**486.2**	**515.8**	**517.7**	143.5	9.35E-28
SAT	714.7	**259.7**	714.7	1022.3	714.7	1080.0	**281.6**	**176.5**	1018.0	1.54E-215
TSP	**572.4**	**583.8**	**621.5**	**649.0**	**618.6**	**675.2**	**622.7**	**620.9**	9.0	2.51E-01
VRPTW	**506.9**	774.8	**481.6**	738.6	**491.3**	834.6	**568.1**	**568.1**	166.7	1.23E-32
KP	666.4	**543.9**	**656.6**	**657.5**	666.4	**624.4**	**573.7**	**575.1**	20.9	4.00E-03
MAC	**554.5**	717.9	**617.5**	668.2	**593.9**	714.5	**556.0**	**541.4**	43.6	2.53E-07
QAP	**648.0**	**553.4**	**541.0**	709.0	**591.3**	707.0	**609.5**	**604.8**	34.0	1.73E-05

Table 4.3: Percentage of each type of move based on the acceptance decision and move delta as improving, equal, and worsening when using the best general move acceptance method for solving an instance, as used in the objective function value traces, of the respective problem domain. Note that no move acceptance mechanism has the ability to reject improving moves. Percentages are reported to 1 decimal place - GD accepted a few moves but these were very rare; hence the reported values of 0.0%

Domain	BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP
Best Method	IE	IE	NA	SARH	IE	TA	AILLA	SARH	TA
Accept $\delta < 0$	0.6%	0.0%	26.9%	10.5%	0.0%	7.8%	34.5%	26.0%	0.3%
Reject $\delta < 0$	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Accept $\delta = 0$	22.9%	3.5%	40.3%	15.0%	0.0%	36.3%	15.5%	4.6%	0.0%
Reject $\delta = 0$	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Accept $\delta > 0$	0.0%	0.0%	16.3%	10.5%	0.0%	8.2%	3.1%	26.0%	0.4%
Reject $\delta > 0$	76.5%	96.5%	16.5%	64.0%	100.0%	47.6%	46.8%	43.4%	99.2%

the normalised results of all BP instances shows that IE is the best general method with a mean rank of 354.3 and that the results of the move acceptance methods do not come from the same distribution. This means that IE must perform significantly better than at least one other move acceptance method.

Post-hoc analysis shows that IE performed significantly better than all other methods apart from TA and SA. This is not surprising since the per-domain tuning setting for TA was $k = 0.0$ meaning that the threshold was set at each iteration as $f(s_i) + 0.0 \times f(s_0) = 0.0$ which is the same mechanism as IE, albeit with a slight computational overhead leading to a reduced performance. Similarly, the per-domain tuned setting for SA has the final temperature set to 0.0 meaning that the cooling schedule reduces the temperature to 0.0 after the first iteration. This again causes SA to behave the same as IE after the first iteration but with the computational overhead.

Since the best move acceptance method for solving instances of the Bin Packing problem was IE, there is not much that can be deduced by way of how the natures of the move acceptance

Table 4.4: Percentage of accepted and rejected moves based on improving, equal, and worsening move deltas when using the best general move acceptance method for solving an instance, as used in the objective function value traces, of the respective problem domain. Note that no move acceptance mechanism has the ability to reject improving moves. For TSP, there was a total of 68 accepted improving moves and 380 accepted equal moves compared to 4908905 rejected worsening moves; hence, the apparent 0.0% of accepted improving and equal moves.

Domain	BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP
Best Method	IE	IE	NA	SARH	IE	TA	AILLA	SARH	TA
Accept $\delta < 0$	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Reject $\delta < 0$	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Accept $\delta = 0$	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Reject $\delta = 0$	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Accept $\delta > 0$	0.0%	0.0%	49.6%	14.1%	0.0%	14.7%	6.3%	37.5%	0.4%
Reject $\delta > 0$	100.0%	100.0%	50.4%	85.9%	100.0%	85.3%	93.8%	62.5%	99.6%

method mechanisms from the taxonomy affect the performance of the local search metaheuristic. The only observation that can be made is that only move acceptance methods that can be manipulated into emulating IE through parameter tuning will perform similar to IE, but with increasing computational overheads negatively affecting their performance.

The progress plot for Bin Packing, shown in Figure 4.3a, shows IE continually improving upon the current solution and finding the best solution at the last step. The progress plot illustrates that the search does not prematurely converge and does not have large plateau regions; hence, IE is most suitable for solving Bin Packing problems, and that methods employing exploration strategies, accepting worsening moves, are unlikely to outperform IE. Observing the acceptance statistics from Table 4.3, we can see that the majority of candidate solutions proposed by the local search method are worsening compared to the solution-in-hand with only 0.6% of moves improving, and 22.9% of moves of equal cost. In conclusion, for solving BP problems under the given framework, it does not appear to be necessary to employ a strategy accepting moves of worsening quality since IE already runs to completion and without converging.

4.2.2 Flow Shop

The best move acceptance method for solving Flow Shop (FS) problems according to the per-domain scores was IE. A boxplot comparison of the performance of the move acceptance methods for solving FS instances can be found in Figure 4.1. The results of performing an ANOVA test on the normalised results of all FS instances shows that IE is the best general method and that AILLA, TA, SA and SARH do not perform significantly different from IE. Similarly to the observations made for solving Bin Packing problem instances, TA, SA, and SARH when tuned per-domain have their settings set such that they behave the same as the IE move acceptance method. AILLA on the other hand only employs a strategy to accept worse quality moves after a period of non-improvement and then reverts back to accepting improving or equal quality moves after an improvement is made. This means that AILLA will accept only a few worse moves for the entire search process due to the ability of the move operators to find better quality solutions most of the time; hence, AILLA on the whole behaves the same as IE.

Likewise, as with Bin Packing, the fact that IE was the best move acceptance method, there is not much that can be said with respect to the natures of the move acceptance methods from the taxonomy and their ability to solve FS problem instances. Move acceptance methods that assume some degree of exploration is necessary to solve a COP and that do not have adaptive parameter control mechanisms that can detect and disable these mechanisms are unlikely to perform well for solving FS given a cross-domain parameter configuration that does not allow for their parameters to be set to extreme values from the start of the search. Hence, from a cross-domain perspective, move acceptance methods with static and dynamic natures of the parameter setting should be avoided, and those with adaptive control mechanisms preferred.

Figure 4.3b shows the progress plot for IE solving the Flow Shop problem. IE is able to improve the best solution found throughout the search; however, the search landscape has many shoulder/plateau regions. The percentage of accepted/rejected moves depending on the move delta are given in Table 4.3. It can be observed that there are much more worse moves being generated using the simple swap neighbourhood operator than there are equal quality moves, and a negligible number of moves resulting in solutions with improved cost. Considering that improving solutions are found after large periods of searching over plateau regions, and the relatively high percentage of equal cost moves that are accepted compared to improving cost moves, a mechanism is needed at the meta-search level, rather than at the move acceptance level, to explore the neighbouring solutions of the solution-in-hand; either to find any potentially improving neighbouring solution(s), or to prevent cycles from occurring on plateau regions.

Looking into the best move acceptance methods for solving each of the FS instances, AILLA was able to find a better solution than IE for 67 out of 155 trials. Taking into account of the mechanism used by AILLA as discussed above, AILLA is able to escape some of these plateau regions faster than IE by chance after accepting a worse quality move. The instantaneous switch back to accepting only equal of improving moves by AILLA means that it is able to outperform other move acceptance methods which employ strategies to accept worse moves of varying degrees throughout the search. In summary, move acceptance methods under the local search metaheuristic framework solving different instances of FS problems benefit from either accepting no to very few worse moves similarly to Bin Packing.

4.2.3 Personnel Scheduling

The best move acceptance method for solving Personnel Scheduling (PS) problems according to the per-domain scores was NA. A boxplot comparison of the performance of the move acceptance methods for solving PS instances can be found in Figure 4.1. The results of performing an ANOVA test on the normalised results of all PS instances shows that NA is the best method for solving instances from this problem performing slightly, but not significantly, better than SA, SARH, TA, and GD when re-tuned per-domain. Looking at the progress plot in Figure 4.3e, we can see that the search performs a random-walk of the search space. The acceptance statistics, given in Table 5.11, shows that the ratio of improving, equal, and worse candidate solutions are fairly balanced at 26.9%, 40.3%, and 32.8% respectively. Analysing the parameter configurations of the move acceptance methods that did not perform significantly different to NA, we can see that:

- SARH - uses a large (99.99%) setting for χ_0 and initiates a reheat after $\frac{1}{1000000}$ of the total computational budget, effectively allowing all moves to be accepted.
- SA - in contrast to SARH uses both high initial and final temperature settings due to the lack of a mechanism to increase the system temperature. SA uses $\chi_0 = 99.99999999\%$ and $\chi_0 = 70\%$ causing it to accept at least 70% of all worse moves for the duration of the search.
- TA - uses a relatively large threshold value of $0.8 \times f(s_0)$ causing it to accept most worse moves.
- GD - uses a setting for $qualityLB = 100$. For smaller instances, this causes GD to accept all moves and performs well, whereas for larger instances less worse moves are accepted and does not perform as good; hence, on average GD performs not significantly worse than NA. Clearly here, an instance-specific parameter tuning method is required for GD to perform well across all instances of the PS domain.

The other move acceptance methods are far more restrictive. IE trivially does not allow worse moves to be accepted and is the worst method for solving PS problem instances. AILLA, and other *non-stochastic basic* move acceptance methods only allow for solutions to be accepted if their objective values are not worse than the objective values of previously visited solutions. Therefore, AILLA is restricted to accepting solutions that are not worse than the objective value of the initial solution and is hence sensitive to initialisation of the search. Furthermore, AILLA reverts to an accept improving or equal strategy after a number of worse moves which is evidently not beneficial for solving PS problem instances.

Non-stochastic threshold move acceptance methods allow worse moves to be accepted based on a threshold that can be calculated in different ways, most commonly with respect to the current solution, or best solution found. AILTA uses the latter strategy, however its internal mechanism for determining these threshold values does not allow it to generate threshold values that are high enough to accept much-worse solutions. AILTA also employs the same strategy as AILLA, reverting to an accept equal or improving moves strategy after improvement is made.

All *stochastic* move acceptance methods evaluated on the PS domain are tuned per-domain to accept the majority of worse moves and perform statistically similar. Since the apparent strategy for solving PS instances is to accept all (or most) moves, the nature of the algorithmic parameter setting mechanism is not useful in determining the success of the move acceptance method when they have been re-tuned per-domain. In the cross-domain perspective however, an adaptive mechanism is needed to control the internal parameters of the move acceptance methods in such a way that worse moves are accepted most of the time.

Due to the implementation of the Personnel Scheduling problem maintaining solution feasibility at all times, it requires the use of a specialised local search move operator to swap shift patterns in the roster which allows for the solution-in-hand to be improved at each iteration. Thus, large and frequent acceptance of worse moves are needed to allow the search to escape regions of optima. In conclusion, a local search metaheuristic simply needs a naïve move acceptance strategy accepting most or all worse moves for solving such problems under the given framework.

4.2.4 Maximum Satisfiability

The best move acceptance method for solving Maximum Satisfiability (SAT) problems according to the per-domain scores was SARH. A boxplot comparison of the performance of the move acceptance methods for solving SAT instances is provided in Figure 4.1. The results of performing an ANOVA test on the normalised results of all SAT instances shows that SARH is the best general method, and that SA and AILLA are equally as good at solving SAT problems. The results show that the nature of the algorithmic parameter setting mechanism has a strong effect on the effectiveness of the move acceptance method to solve MAX-SAT problem instances with the move acceptance methods utilising a static algorithmic parameter setting mechanism performing poorly on this problem. The nature of the accept/reject decision did not have much of an effect on the per-domain performance of algorithms to solve SAT problems.

SA and SARH use initial and final temperature settings of $\chi_0 = 92\%, \chi_0 = 1/64\%$ and $\chi_0 = 50\%, \chi_n = 1/640\%$ respectively. The difference in parameter settings is despite the *wait_time* setting of SARH being as high as 90% of the total search time and can be attributed to the sensitivity of interactions between parameters with a fractional DOE. This leads to SARH appearing to outperform SA, although in practice both algorithms behave the same when *wait_time* is sufficiently large. AILTA is the only method to break this pattern because of its internal mechanism for calculating threshold values not allowing it to use threshold values large enough for this problem. AILTA could therefore not accept worse moves, leading to its inferior performance and similarity in performance to IE. Neither IE, TA, nor AILTA could accept worse moves and it is evident from the results that this causes them to perform poorly. NA accepts far too many worse moves resulting in overall performance much worse than those that accept no worse moves. The performance of GD is extremely variable. This is due to the more-general per-domain parameter tuning which leads it to perform well on some instances but bad on other instances where a reasonable target value (*qualityLB*) is different to that determined from the tuning procedure.

The progress plot, shown in Figure 4.3f, shows SA(RH) solving a Maximum Satisfiability problem. The behaviour shown is characteristic of Simulated Annealing algorithms, initially accepting many worse moves, and reducing them over time to converge on a good quality solution towards the end of the search. In this scenario, SARH behaves the same as SA (without reheating) with the setting of 90% for the *wait_time* parameter used for solving this problem effectively disabled the reheat mechanism. The acceptance statistics, given in Table 5.11, shows that the number of worse candidate solutions is high at 74.5% with only 10.5% of all candidate solutions being improving, and the remaining 15.0% equal cost. In conclusion, a strategy to accept some worse moves is needed for solving Maximum Satisfiability problems, however this needs to be *controlled* appropriately, either as a dynamically or adaptively, as the naïve acceptance method performs the worst out of the move acceptance methods investigated.

4.2.5 Travelling Salesman Problem

The best move acceptance method for solving Travelling Salesman (TSP) problems according to the per-domain scores was IE. A boxplot comparison of the performance of the move acceptance methods for solving TSP instances can be found in Figure 4.2. The results of performing an ANOVA test on the normalised results of all TSP instances shows that IE is the best general method, but with all other move acceptance methods performing not significantly different. While it would appear that nothing much can be said about the effectiveness of the different move acceptance methods, we must consider the parameter settings and control mechanisms used in the evaluated move acceptance methods.

- TA - the threshold was set to $f(s_i) + 0.001 \times f(s_0)$ meaning that very few worse moves are accepted for large problem instances, and no worse moves are accepted for smaller instances.
- GD - $qualityLB = 100,000$. With respect to some larger problem instances, this is very small meaning that GD behaves like IE after a number of iterations, whereas for some smaller instances this is too high causing GD to behave as an accept all moves strategy. This means that GD performs well for solving larger instance sizes, accepting IE moves, but relatively poorly for smaller instances where GD turns in to random walk.
- SA - The final temperature was tuned to $\chi_n = 0\%$ causing SA to behave the same as IE.
- SARH - The final temperature was tuned to $\chi_n = 0\%$ causing SARH to behave the same as IE.

Both AILLA and AILTA only accept worse quality solutions if there is a period of no improvement and immediately return to accept equal or improving moves after an improvement is made; hence, their performance is not too dissimilar from IE given the computational overhead of their adaptation mechanisms. While NA does not perform significantly worse than IE, it is the worst performing move acceptance method. This can be due to the differences between the problem instances meaning that those tuned to accept no worse moves perform poorly on other problem instances. Hence, in retrospect, NA is able to solve such instances better due to its ability to accept worse moves.

Figure 4.3c provides the progress plot for IE solving a TSP problem. It can be seen that the solution-in-hand is improved over time along with the best solution found suggesting that no strategy for accepting worse moves is need for solving TSP instances well. This observation is made based on the improved per-domain performance scores when using IE over strategies that always use exploration strategies such as GD and NA, despite the differences being insignificant. The acceptance statistics, given in Table 5.11, shows that the ratio of improving, equal, and worse candidate solutions are extremely skewed to worse candidate solutions, generating (to one decimal place) 100.0% of all solutions. Note that there were actually 68 improving candidate solution generated compared to 4908905 worse candidate solutions. In conclusion, while statistically insignificant for solving TSP problems, the same observations are made as with Bin Packing and Flow Shop

Scheduling problems where a move acceptance method should accept no (or very few) worse moves for it to perform better than other methods.

4.2.6 Vehicle Routing with Time Windows Problem

The best move acceptance method for solving Vehicle Routing (VRPTW) problems according to the per-domain scores was TA. A boxplot comparison of the performance of the move acceptance methods for solving VRPTW instances is shown in Figure 4.2. The ANOVA test on the normalised results of all VRPTW instances shows that TA is the best general method, with TA, AILTA, IE, SA, and SARH performing not significantly different.

The progress plot shown in Figure 4.3h shows TA's ability to improve the best solution over time, and without accepting worse moves of large magnitude which would otherwise cause the search to worsen over time. The acceptance statistics, given in Table 5.11, shows that only 7.8% of all candidate solutions are improving with over half of all candidate solutions having worse objective quality. IE, accepting no worsening moves, also performs well on this domain, although its performance on the whole is worse than TA suggesting IE becomes stuck in a local optima. SA and SARH are tuned per-domain such that their final temperatures are set very small, $\chi_n = 1/64\%$ for SA and $\chi_n = 0\%$ for SARH which causes both methods to either very quickly, or instantly reduce the system temperature such that no worse moves can be accepted. This means that SA and SARH perform similarly to IE but with reduced performance due to their computational overhead. AILTA on the other hand performs better than IE, and only slightly worse than TA. AILTA only accepts improving or equal moves until a given computational budget has passed without improving the solution, at which point, it uses an internal mechanism to calculate a threshold value (similar to TA but with respect to the best solution rather than the solution-in-hand). Similarly to the above problem domains, the performance of NA is very bad, accepting too many worse moves when such an aggressive exploration method is not needed; furthermore, the performance of GD is extremely variable due to the more-general per-domain tuning method giving a target threshold which is not specific to each instance being solved.

The nature of the accept/reject decision has the most significant effect on the algorithm's performance with non-stochastic threshold methods performing well due to their ability to allow a number of worsening moves with small move deltas to be accepted. This is in contrast to non-stochastic basic methods which can only re-use the objective values of previously visited solutions as threshold values which may be too high. In conclusion, an acceptance strategy needs to be able to accept worse moves to most effectively solve VRPTW problems, but the threshold values used to determine acceptance of such moves need to be carefully determined since if they are too high, the search may not converge, and if they are too low, they will perform on par with IE.

4.2.7 0-1 Knapsack Problem

The best move acceptance method for solving 0-1 Knapsack problems (KP) according to the per-domain scores was AILLA. A boxplot comparison of the performance of the move acceptance methods for solving KP instances can be found in Figure 4.2. The results of performing an ANOVA test on the normalised results of all KP instances shows that AILLA is the best general method, with SA, SARH, NA, TA, and GD performing equally as well. To understand the behaviour of the move acceptance methods for solving KP problem instances, we must observe their per-domain tuned parameter settings:

- SA - SA uses $\chi_0 = 80\%$ and $\chi_n = 1/32\%$ which means it is able to accept a large number of worse moves throughout the search, reducing over time.
- SARH - Similar to when SARH was used for solving SAT problems, the wait_time parameter setting is set very high (0.9999) meaning that the reheat mechanism will never be activated and is not useful in this case. The initial temperatures were tuned to the same $\chi_0 = 80\%$ however the final temperature was set slightly higher at $\chi_n = 0.25$ due to the fractional DOE. The behaviour of SARH is therefore the same as SA.
- TA - The threshold value for TA was set to be $\tau = 1 \times 10^8$ which is higher than the upper bound of the objective function for KP (1) meaning that all moves are accepted.
- GD - The target value of the threshold for GD was set to be $qualityLB = 0$, and this is equal to the lower bound of the objective function for KP. In practice, this allows for the search to accept many worse moves, but decreases the magnitude of those accepted linearly towards the end of the search, behaving similar to SA.

NA by definition accepts 50% of all worse moves. The progress plot for AILLA solving an instance of the KP problem is illustrated in Figure 4.3d. Here, we can see that AILLA is in fact allowing many worse moves to be accepted, appearing to behave like a random walk. The acceptance statistics, given in Table 5.11, shows that the ratio of improving and equal to worse candidate solutions are fairly balanced at 50% each. AILLA accepts all equal and improving moves, accepting 6.3% of all worse moves, and rejecting the remaining 93.8% of them. Both move acceptance methods which perform significantly worse are IE and AILTA. IE is unable to accept any worse moves, whereas AILTA is unable to accept moves worse enough to benefit the search.

The nature of the accept/reject decision of the move acceptance method clearly has a significant effect on the effectiveness of a local search metaheuristic with all *stochastic* move acceptance methods performing well with the adaptation mechanism within AILLA allowing it to perform well also. Complete randomness of the acceptance of worse moves results in a degraded performance of the move acceptance method, and we can see that in general *dynamic* and *adaptive* natures of the parameter setting mechanisms perform better than those with *static* methods since they can control the degree to which worse moves are accepted. It should be noted that the greedy initialisation procedure used in this domain means that it is an absolute requirement that moves worse than the initial solution are accepted under such framework as the search starts from a local optimum. Moreover, the simple heuristic operators *pack random* and *remove random* means that once the knapsack becomes too full to add any other items, an item must be removed before another can be added. This however means that the objective quality of the solution-in-hand must worsen (increase) before another item can be packed to improve the solution. Hence, the observations made here are not unsurprising. In conclusion, a move acceptance method must accept a high number of worse moves initially, but must decrease over time to perform well with move acceptance methods tuned to accept all moves performing relatively poorly, and those which by design accept no worse moves performing significantly worse.

4.2.8 Max Cut Problem

The best move acceptance method for solving Max Cut problems (MAC) according to the per-domain scores was SARH. A boxplot comparison of the performance of the move acceptance methods for solving MAC instances can be found in Figure 4.2. The results of performing an ANOVA test on the normalised results of all MAC instances shows that SARH is the best general method, with SA, IE, TA, and AILTA performing equally as well. The ranks from the ANOVA test shows that SARH only very slightly outperforms IE and SA. Due to the setting of $\chi_n = 0$ for SA, SA behaves the same as IE after the first iteration with computational overheads, thus both perform similarly as well. SARH however does not have a zero final temperature and allows it to behave in the traditional way. The same as with other domains where SARH performs well, the fractional DOE means that the final temperature is different between both SA and SARH despite the wait_time parameter of SARH being set sufficiently high to disable the reheat mechanism. This allows for SARH to accept worse moves, as expected by SA methods, and is shown in Figure 4.3g for solving MAC instance #5. We can see that for solving this instance, SA(RH) accepts a large number of worse moves and is unable to improve the best solution until the last 1/4 of the computational budget. It should be noted though that a high setting for χ_n in one instance may not be as beneficial for solving a different instance of the same problem. Hence, the resulting parameter settings using the more-general per-domain tuning approach can result in substandard settings for unseen problem instances. The acceptance statistics, given in Table 5.11, shows that the majority of candidate solutions are of worse quality, constituting 69.4% of all candidate solutions. 26.0% of all candidate solutions are of improving quality. SA(RH) accepts 37.5% of all worsening moves throughout the search process which is approximately twice as high as than for solving SAT problem instances, but comparing the ratio of improving candidate solutions between MAC and SAT, MAC also produces 2.5 times as many improving candidate solutions. TA and GD both use relatively high/low threshold settings of $\tau = 5 \times 10^{-4}$ and $qualityLB = 1 \times 10^{-7}$ respectively for solving the instances of the MAC problem. This again is due to the per-domain level of tuning and shows that per-instance tuning is required by these methods for them to perform well.

The nature of the algorithmic parameter settings had a marked effect on the effectiveness of the move acceptance methods for solving this problem with *dynamic* and *adaptive* parameter control methods performing better in general than *static* methods. In conclusion, while IE performs fairly well for solving MAC problem instances, a move acceptance method needs a mechanism to be able to accept some worse moves to perform better, but accepting too many worse moves with a high delta value has a detrimental affect on the performance of a local search metaheuristic embedding the move acceptance method to perform well.

4.2.9 Quadratic Assignment Problem

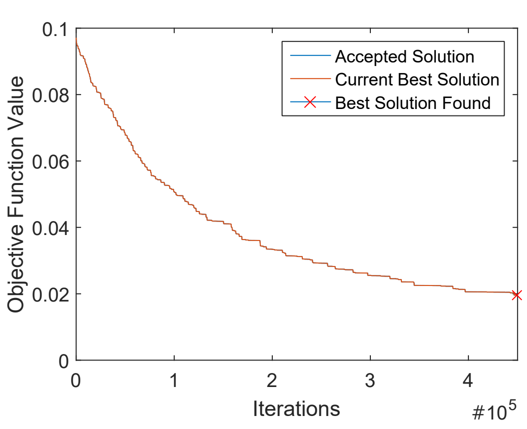
The best move acceptance method for solving Quadratic Assignment (QAP) problems according to the per-domain scores was TA. The results of performing an ANOVA test on the normalised results of all QAP instances shows that TA is the best general method, with IE, AILLA, AILTA, SA, and SARH performing equally as well. Figure 4.3j shows TA solving instance #0 of the QAP problem. We can see that the threshold, τ, is only slightly higher than the solution-in-hand and this allows for enough worse moves to be accepted such that the best solution is continually improved over time without descending into a random walk of the search space. The acceptance statistics, given in Table 5.11, shows that the majority of candidate solutions are of worse quality (99.6%), compared to just 0.3% of all candidate solutions that are improving. On the whole, TA accepts only 0.4% of all generated worse candidate solutions. IE is the worst performing move acceptance method of those that do not perform significantly different to TA and illustrates that some degree of acceptance of worse moves is required for a move acceptance method to perform well. Such mechanisms are present in AILLA and AILTA where worse moves are accepted after a period of non-improvement before returning to accepting equal or improving moves. SA and SARH both have sensible settings for their initial ($\chi_0 = 70\%$ and 60% respectively) and the same final temperature ($\chi_n = 1/64\%$) with SARH using a sensible wait_time setting which allows for the re-heating mechanism to be activated.

NA permits too many worse moves and does not discriminate them based on the magnitude of how worse the moves are leading to consistently poor performance. GD, using its per-domain parameter configuration, has the opposite problem where the threshold value is decreased so fast, that it quickly is unable to accept any worse moves, and due to its definition, accepts only improving moves that have objective values greater than the current threshold value. This means that in this case, GD is unable to accept equal quality moves lending to its poor performance.

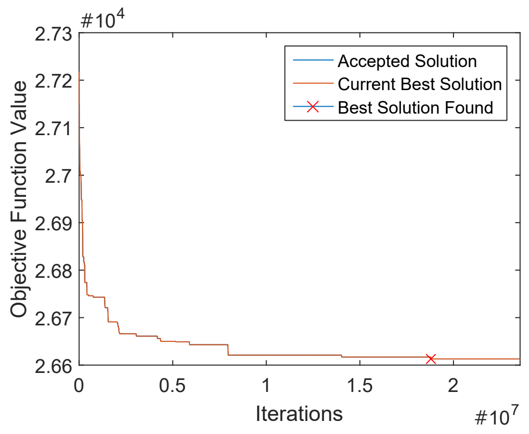
The nature of the parameter settings and nature of the accept/reject decisions did not significantly affect the ability of a move acceptance method to perform well for solving QAP instances. In conclusion, the determining factor for a move acceptance method to perform well in this domain was for it to be able to accept equal moves, and worse moves with small delta values with those methods permitting large moves deltas performing significantly worse.

4.2.10 Per-domain Performance Observations

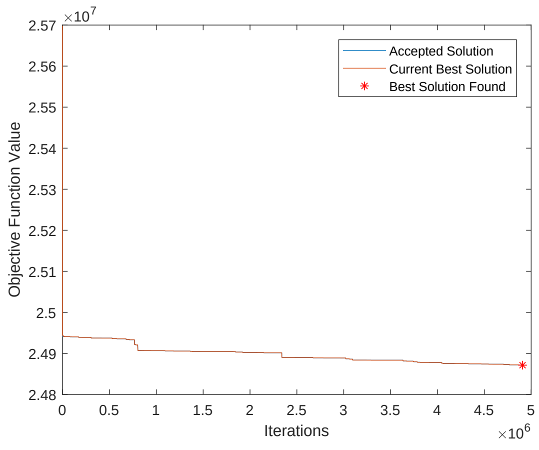
To observe the best move acceptance method(s) for solving each type of problem, a Wilcoxon Signed Rank test is performed between the best general move acceptance method, as determined by the best per-domain score, and the remaining methods for each problem using the normalised set of results from all 5 instances. There are some domains where a single move acceptance method performs statistically significantly better than all other methods, and some domains where multiple move acceptance methods perform not significantly different from the best general method, hence a set of such methods are said to perform equally as well. Bin Packing (IE), Personnel Scheduling (NA), MAX-SAT (SARH), Vehicle Routing with Time Windows (TA), 0-1 Knapsack (AILLA), Max Cut (SARH), and Quadratic Assignment (TA) are all solved best by a single move acceptance method as indicated in parenthesis. For these cases, it is interesting to see that the different move



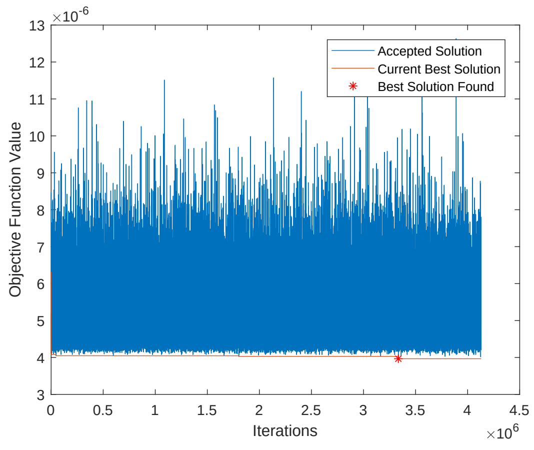
(a)



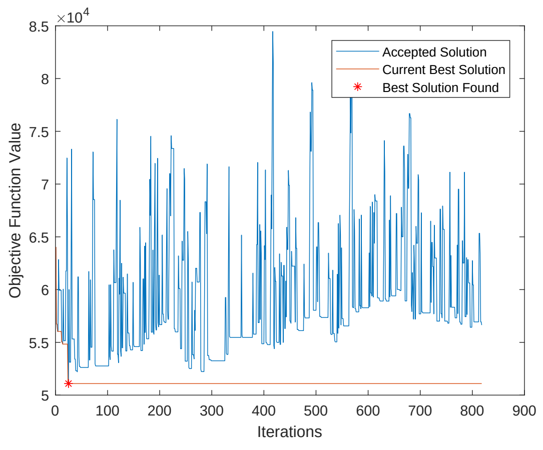
(b)



(c)

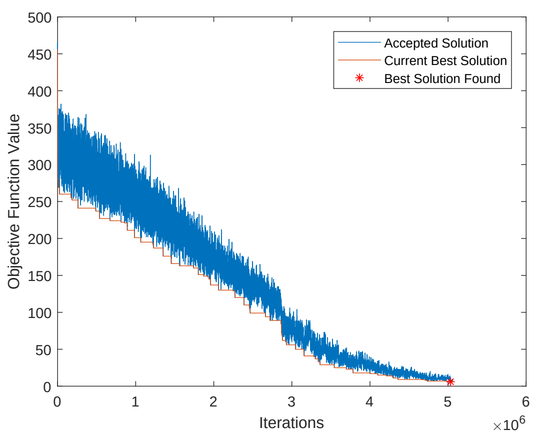


(d)

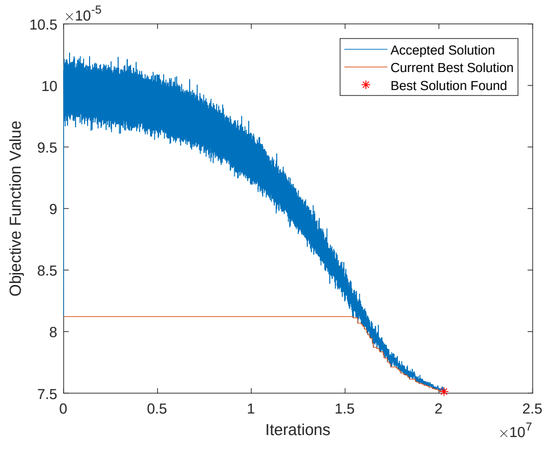


(e)

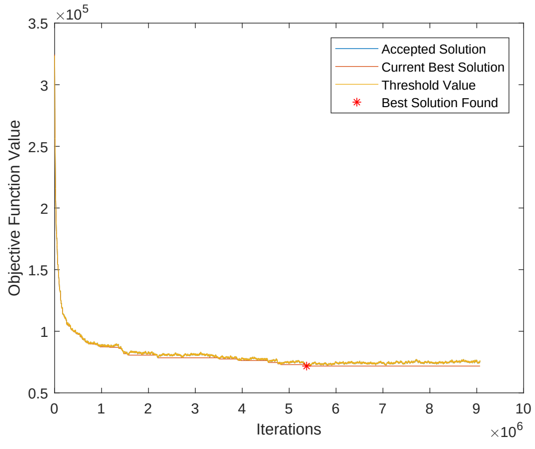
Figure 4.3: Objective function value traces of accepted and current best solutions given a 10 nominal minute computational budget for: IE solving (a) instance 11 of the Bin Packing problem, (b) instance 11 of the Flow Shop problem, and (c) instance 8 of the Travelling Salesman Problem, and AILLA (d) solving instance 3 of the 0-1 Knapsack Problem, and NA (e) solving instance 9 of the Personnel Scheduling problem. Note that the threshold values have been disabled for AILLA due to frequent switching between accepting all moves, and moves proportional to the best in the current stage causing the plot to be illegible.



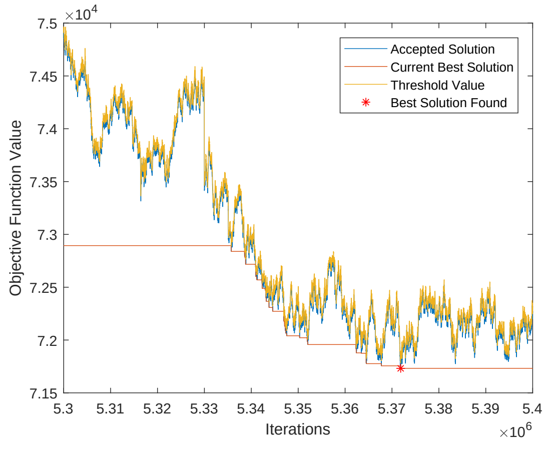
(f)



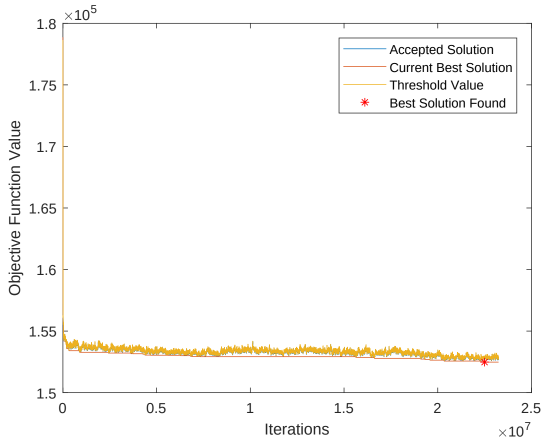
(g)



(h)



(i)



(j)

Figure 4.3: Objective function value traces of accepted solutions, current best solutions, and acceptance thresholds given a 10 nominal minute computational budget for: SARH solving (f) instance 5 of the MAX-SAT Problem and (g) solving instance 5 of the Max-Cut Problem, and TA solving (h) instance 6 of the Vehicle Routing with Time-Windows problem, with (i) being zoomed in to illustrate the difference between the accepted solution fitness and threshold values, and TA also solving (j) instance 0 of the Quadratic Assignment Problem.

acceptance methods come from each classification of the accept/reject nature from the taxonomy, and that the settings of their parameters were sensible rather than extreme values that cause their intended search strategy to be lost. Flow Shop (IE, AILLA), and Travelling Salesman (IE, AILLA) are all best solved by a group of move acceptance methods given in parenthesis proceeding the respective domains. The set of move acceptance methods which best solves each problem are all from the *basic* setting of the nature of the accept/reject decision. Here, IE does not perform significantly better than AILLA since the mechanism for AILLA to accept worse moves depends on a consecutive number of non-improving iterations to be performed. AILLA does however perform better than IE on some Flow Shop and Travelling Salesman problem instances, and this is due to IE spending a significant amount of its time on plateau regions in the search space, potentially getting stuck at local optima. During this time, AILLA allows a worse move to be accepted allowing the search to escape such regions.

In summary, move acceptance methods with different natures of the accept/reject decision are needed for solving different COPs. Some problems, such as BP, FS, and TSP, are solved well by accepting only improving or equal cost moves, whereas other problems require worse moves to be accepted. Some problems benefit from accepting worse moves with small differences between the objective values of the current and candidate solutions, including VRPTW and QAP, whereas others prefer accepting large and frequent worse cost moves, such as PS and KP. The SAT and MAC problems required a balance between accepting large and small move deltas, reducing the magnitude of worse moves that are accepted in time. Some of these preferences can be attributed to the relationship between the solution representation and move operators being used and their objective functions, even under such a single point-based stochastic local search metaheuristic framework. KP, for example, uses a binary representation with two move operators to remove or add items from or into the Knapsack. Once the knapsack is full, clearly items must be removed from it to try other items; however, this is not possible without accepting solutions that have worse cost since removing an item will result in a worse profit. SAT is similar in that the representation used is the same, however the move operator is just a simple bit-flip operator, negating the truth value of a random variable. Once an assignment is reached such that no further single bit flips can improve the cost of the solution-in-hand, the search will become stuck; hence, worse moves need to be accepted. The nature of the SAT problem however means that one single bit flip can affect the truth value of multiple clauses; hence, several bit flips after accepting a worse move may be needed to improve the solution, or even consecutive bit flips may be able to improve the solution-in-hand. Therefore, accepting worse moves should not be indiscriminate.

Another interesting observation to make is that the best move acceptance method changes between each domain depending on the ratio of improving, equal, and worse candidate solutions generated from the solution-in-hand. Domains that generate a high number of worse candidate solutions and low number of improving candidate solutions (BP, FS, TSP, VRPTW, and QAP) were best solved using either IE, or TA with a small but non-zero setting for τ. Domains that generated a comparatively high ratio of improving candidate solutions (PS, SAT, KP, and MAC) were solved well using *stochastic* move acceptance methods.

If a local search metaheuristic is sought for solving problems from a particular domain under the local search metaheuristic framework, then the respective move acceptance method is recommended to be used. If a *new* or *unknown* problem is to be solved by a *practitioner* who may want a cheap and fast solution to their problem, and they have time to re-tune its parameters, we showed that out of those tested, Simulated Annealing was the most effective method *in general* with the best cross-domain μ_{norm} score and is therefore recommended for such scenarios. SA has the ability when tuned to behave very differently based on its parameter settings allowing for it to accept no worse moves at all ($\chi_n = 0$), all or most worse moves when the initial and final temperatures are set high, or a balance between the two when the initial temperature is high and the final temperature is small (but not equal to 0). In turn, this allows for the practitioner to only need to consider the single move acceptance method since it can exhibit all required behaviours for solving different problems when re-tuned.

4.2.11 Cross-domain Results and Observations

The cross-domain scores for the local search metaheuristic embedding the different move acceptance methods are shown in the final column of Table 4.1 as the sum of per-domain scores over nine problem domains. Each per-domain has a theoretical minimum and maximum score of 0 and 45 (9 domains $\times$ 5 instances per domain) respectively (lower is better). The results show that most move acceptance methods, with GD tuned per-domain and NA being the exception, perform better than the theoretical average; that is, obtaining scores less than 22.5. IE as a move acceptance method represents a standard hill climbing local search approach with no mechanism for accepting worse moves, and methods failing to surpass its performance being deemed poor for solving the cross-domain problem. Moreover, IE is parameter free meaning that no expert intervention is required to reconfigure it for solving different problems. TA, SA, and SARH all improved over IE's cross-domain performance when re-tuned per-domain whereas AILLA, AILTA, GD, and NA performed worse under the given local search metaheuristic framework using the high-level per-domain parameter configurations.

No move acceptance method could consistently perform the best across all problems, despite their parameters being re-tuned for each problem domain where necessary. Moreover, different move acceptance methods were able to outperform each other when solving different problem instances for some of the domains. This result highlights the trade-off between requiring and selecting from multiple algorithms which perform exceptionally for solving a few problems each, and having a single algorithm which performs sufficiently well for solving all problems.

The cross-domain performance of SARH closely followed that of SA with scores of 10.88 and 8.62 respectively. An important factor in the design of general-purpose search methods is the ideological goal of eliminating the need for an expert to reconfigure them when aimed at solving different problems. That is, once the search method has been designed, it can be used as-is for solving any other problem. In this study, the parameters of the move acceptance methods were re-tuned for each problem domain to show their potential as a general-purpose search method requiring no expert intervention to re-tune them for solving different instances of the same problem.

Generally, it can be observed that using algorithmic parameter setting mechanisms which are not static improves the cross-domain performance of the algorithms at the same accept/reject nature level. For static methods, use of a non-stochastic basic algorithmic parameter setting mechanism has a clear advantage over a non-stochastic threshold mechanism which in turn outperforms a stochastic mechanism since there is no way for them to adapt to different characteristics of different instances from the same problem. This is no surprise since static parameter setting mechanisms have no functionality to reduce the number of accepted worse moves if too many are being accepted. In contrast, this is now possible given dynamic or adaptive parameter setting mechanisms where threshold and stochastic mechanisms can improve over move acceptance methods using only static control methods. It is extremely important to carefully consider the design of adaptive algorithmic parameter setting mechanisms. Just because a move acceptance method controls their internal parameters does not necessitate that it will perform well across different problems. As an example, AILLA and AILTA are classified as adaptive, and their adaptation mechanisms are quite similar acting upon a set of values which are used directly to determine the acceptance threshold. AILTA acts upon a set of threshold values (ϵ) which are multiplied with the cost of the best solution found so far whereas AILLA acts upon a set of previous best solution values. The significant difference between AILTA and AILLA is that the set of threshold values used by AILTA are themselves fixed within the logic of the control mechanism whereas AILLA uses a set of previous best solution values that depend on maintaining a memory of the search history and hence reflect feasible and useful threshold values as learnt from the problem instance being solved. For this reason, AILTA has a poor cross-domain performance compared to AILLA. This observation is important in the context of cross-domain search since the ranges of objective values from different objective functions and problem instances can vary significantly. Threshold values in move acceptance methods using a *non-stochastic* nature of the accept/reject decision must be derived from the instance being solved rather than being defined as a part of the algorithm design process.

4.3 On the Effectiveness of Parameter Tuning versus Choice of Move Acceptance Method for Cross-domain Search

It is well known that the parameters of metaheuristics require tuning for them to perform well - even at a per-instance level. Hence, parameter tuning algorithms exist that operate on a per-instance basis, such as instance specific parameter tuning methods [27] that suggest parameter settings based on specific characteristics/features of the problem instance being solved. The ultimate aim of cross-domain search is to devise a single high-level search method that does not require an expert to re-configure it to perform well for solving real-world COPs; that is, in the absence of such parameter tuning algorithms/algorithm configuration methods after the search method has been designed.

One of the goals of cross-domain search is to *reduce* and *preferably eliminate* the need for an expert to configure search methods for them to perform well. The per-domain tuning approach can be seen as *reducing* the requirement for expert intervention of an algorithm's configuration, while the cross-domain approach *eliminates* the requirement for such expert intervention after the algorithm's design phase. Since the focus of this work is on cross-domain parameter tuning, the influence of two different "more-general" tuning approaches (per-domain vs. cross-domain at the high level) with the goal of reducing the overall expert intervention are compared. There has not been a study before which investigates the effectiveness of these parameter tuning approaches under a *cross-domain search* framework, and hence the effects that these approaches have on the cross-domain performance of the local search metaheuristic framework is explored using the move acceptance methods. Considering that the cross-domain performance of the local search metaheuristic is also significantly affected depending on the choice of move acceptance method, in this section the cross-domain effectiveness of parameter tuning using the two parameter tuning approaches considered for cross-domain search methods is compared to that of the choice of the move acceptance method itself.

The cross-domain performance of the four move acceptance methods that have both per-domain and cross-domain parameter configurations are used in this work. That is, Threshold Accepting, Great Deluge, Simulated Annealing, and Simulated Annealing with Reheating. Section 4.3.1 discusses the results comparing the effects of the parameter tuning approaches on the cross-domain performance of the local search metaheuristic embedding each move acceptance method. Section 4.3.2 compares the significance in the difference in cross-domain performance of the local search metaheuristic based on the choice of the move acceptance method considering both per-domain and cross-domain parameter configurations. The full set of results for each move acceptance method — parameter tuning configuration pairs is available online at: `http://dx.doi.org/10.13140/RG.2.2.14307.99364` along with their associated $f_{norm}(s)$ values for the parameter tuning effects and boxplots.

4.3.1 Results for Parameter Tuning Effects

A one-tailed Wilcoxon Signed Rank test ($\alpha = 0.95$) was performed on each per-domain and cross-domain tuned move acceptance method pairs using their $f_{norm}(s)$ scores calculated for each pair of move acceptance method tuning approaches from all 45 problem instances. The results, given in Table 4.5, show that for the TA and SARH move acceptance methods, per-domain tuning statistically significantly improved its cross-domain performance compared to cross-domain tuning with p-values of 2.444×10^{-204}, and 0. On the other hand, cross-domain tuning significantly improved the performance of SA compared to per-domain tuning ($p = 0.01038$). One reason why this could be the case is due to over-tuning of SA's parameter settings on certain tuning instances which then perform poorly for characteristically different instances from the same problem. The cross-domain level of parameter tuning is obtained from tuning SA on instances from multiple domains and hence a general parameter configuration is produced which is not over-tuned to a particular problem instance. There is no significance between the two parameter tuning levels for GD since the setting for *qualityLB* needs to be derived specific to each instance. Any general setting for this is bound to lend to poor performance overall due to changes in objective functions and problem instance characteristics.

The implications that these results have is that for some of the existing move acceptance methods, increased expert intervention in the form of parameter tuning is required to improve their cross-domain performance, whereas other move acceptance methods can benefit from more-general tuning approaches such that they are not over-tuned to specific instances of a given problem to be solved. A purely cross-domain tuning approach, i.e. a single parameter configuration which is re-used for each problem domain and instance, which is essential for eliminating the need for expert intervention, significantly reduces the cross-domain performance of the local search metaheuristic using some of the existing move acceptance methods.

Table 4.5: Comparison of per-domain tuning versus cross-domain tuning effects on the cross-domain performance of move acceptance methods under a local search metaheuristic framework. Note that μ_{norm} and ν_{norm} scores cannot be compared between different move acceptance methods as they are re-calculated for each (per-domain - cross-domain) move acceptance pair.

Move Acceptance Method	$\sum \nu_{norm}$	Mean	Std.	Sgn.	μ_{norm}	Mean	Std.	p-value
	per-domain				cross-domain			
TA	422.27	0.303	0.328	$<$	1028.64	0.737	0.290	2.444×10^{-204}
GD	875.37	0.628	0.389	$\leq$	876.75	0.628	0.387	3.163×10^{-1}
SA	697.55	0.500	0.316	$>$	651.21	0.467	0.340	1.038×10^{-2}
SARH	439.84	0.315	0.262	$<$	909.61	0.652	0.249	0

4.3.2 Results for Choice of the Move Acceptance Method Effects

The cross-domain performance of the move acceptance methods that are tuned cross-domain (and in a second test that are tuned per-domain) were evaluated using a Friedman test with repeated measures (reps=31, n=45) at a confidence interval of 95%. The results of both cross-domain and per-domain tests can be found in Table 4.6. The results shows that there is no statistically significant difference in the cross-domain performance of a single point-based stochastic local search metaheuristic depending on the chosen move acceptance method, both when tuned cross-domain ($p = 0.205, \chi^2 = 4.58$), and per-domain ($p = 0.150, \chi^2 = 5.31$).

These results therefore show that when using higher-level parameter tuning methods, either per-domain or cross-domain, the choice of move acceptance method does not significantly affect the cross-domain performance of a local search metaheuristic. Hence, the design of a move acceptance method, and its non-reliance on instance specific parameters, is important if we want to be able to use if for cross-domain search.

4.3.3 Observations of parameter tuning and choice of the move acceptance method effects

The results of the parameter tuning effects and the choice of the move acceptance method effects experimentation shows while it is possible to tune the move acceptance methods for cross-domain

Table 4.6: Friedman test comparing the cross-domain performance of the move acceptance methods using cross-domain and per-domain parameter tuning configurations with n_0 that all results are from the same distribution at CI = 95%. The values are the mean ranks (lower is better) of the aforementioned test. The best move acceptance method (as chosen as that with the lowest mean rank), and those which do not statistically significantly differ from the best, for each domain being stylised bold.

Parameter Configuration	TA	GD	SA	SARH	$\chi^2(3)$	p
Cross-domain	**2817.9**	**2846.0**	**2724.0**	**2774.1**	4.58	2.05×10^{-1}
Per-domain	**2770.7**	**2875.7**	**2750.9**	**2764.6**	5.31	1.50×10^{-1}

search, their cross-domain performance can vary significantly depending on the exact move acceptance method. Rather than the tuning of parameters being important, at the perspective of cross-domain search, the design of the move acceptance methods themselves become important.

A boxplot of the $f_{norm}(s)$ scores for both parameter tuning approaches for all four move acceptance methods can be found in Figure 4.4. The boxplot illustrates that while not significant, the differences in the cross-domain performance of each move acceptance method at the same parameter tuning level (i.e. cross-domain tuning and per-domain tuning) varies, albeit not significantly, and that with the exception of SA, the cross-domain performance of per-domain tuned move acceptance methods can improve slightly over the cross-domain tuning approach for the same move acceptance method.

The findings from these experiments would therefore suggest that the design of a more effective move acceptance should have a higher research priority over parameter tuning, at least in the context/field of cross-domain search, for the following three main reasons:

1. The existing move acceptance methods are extremely sensitive to parameter configurations, irrespective of whether they have a good or poor cross-domain performance, meaning that it is unlikely for an existing method to perform well as a cross-domain search method without significant parameter tuning efforts - this goes against a key goal of cross-domain search which is to eliminate the need for expert intervention after such a search method has been designed.

2. The choice of move acceptance method has a significant effect on the cross-domain performance of a local search metaheuristic.

3. Move acceptance methods that have been re-tuned for each domain can perform significantly worse than "better" move acceptance methods that have only been tuned cross-domain, and vice versa.

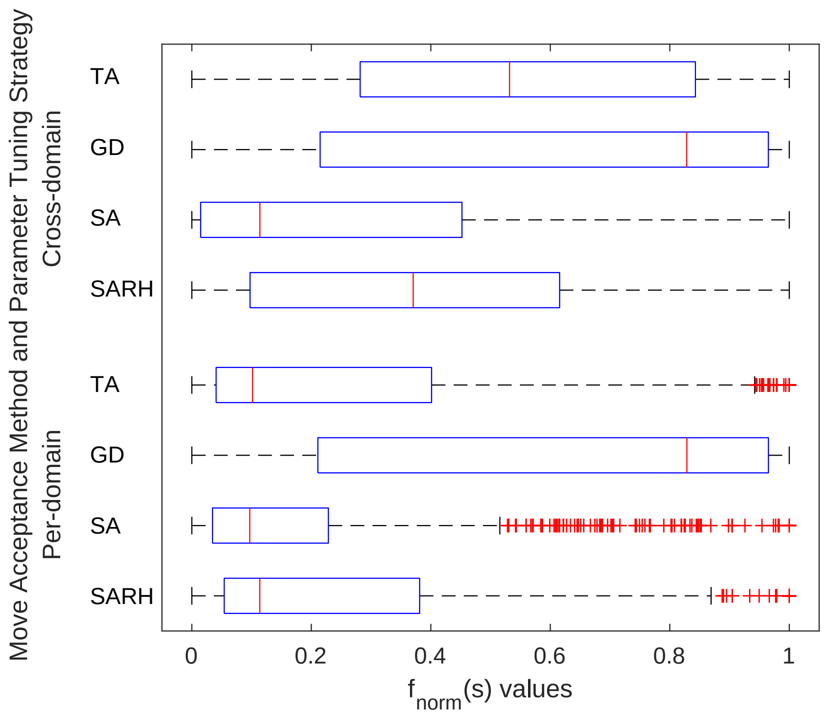


Figure 4.4: The $f_{norm}(s)$ scores calculated over all eight "move acceptance method - parameter tuning approach" pairs showing the cross-domain performance differences between each move acceptance method at the same parameter tuning approach level, and between each move acceptance method using both cross-domain (top) and per-domain (bottom) tuning approaches. '+' marks symbolise statistical outliers according to either $f_{norm}(s) > q_3 + 1.5 \times (q_3 - q_1)$ or $f_{norm}(s) < q_1 - 1.5 \times (q_3 - q_1)$ where q_1 and q_3 are the 25th and 75th sample data percentiles.

4.4 Summary

In this chapter, an empirical study was conducted to investigate the effects that move acceptance methods, as components of single-point based stochastic local search metaheuristics, have on the cross-domain performance of such algorithms for solving multiple combinatorial optimisation problems. The experimental results across a benchmark of nine different computationally hard problems highlight the shortcomings of existing and well-known methods for use as components of cross-domain search methods, despite being re-tuned for solving each domain. These include such methods that rely on their parameters to be set specific to the instance being solved, such as *qualityLB* for GD, the inability for non-stochastic basic move acceptance methods to accept solutions whose objective values are worse than the initial solution, the assumption of some methods that exploration is a requirement for good performance when in some cases (BP, FS, TSP) this is not the case under the given framework, for example all static and dynamic move acceptance methods with a non-stochastic threshold or stochastic nature of the accept/reject decision embed a strategy to allow worse moves that cannot be disabled on-the-fly by an adaptive control mechanism.

The work in this chapter was then extended to observe the effects of parameter tuning versus the choice of the move acceptance method on the cross-domain performance of a local search metaheuristic. The results of which showed that both parameter tuning approaches, and the choice of the move acceptance method have a significant effect on the cross-domain performance of a local search metaheuristic, but at the cross-domain level of parameter tuning, the effects of using different move acceptance methods becomes more significant that at the per-domain tuning level. As one of the goals of cross-domain search is to design a single solution method that does not require expert intervention to perform well, it is evident that a single point-based stochastic local search metaheuristic that makes use of the move acceptance methods evaluated in this study are unlikely to perform well unless their parameters are re-tuned for each and every problem instance.

The outcome of this study suggests that to advance the cross-domain performance of single point-based stochastic local search metaheuristics, new ideas need to be generated for the design of efficient move acceptance methods which are both operable, and effective for solving problems, across multiple domains well using only a single cross-domain parameter configuration. The following chapter proposes a new move acceptance method that is designed to be used for cross-domain search using a single parameter configuration while having a cross-domain performance that is as-good-as the existing move acceptance methods that are re-tuned for each problem domain.

Chapter 5

The History-based Adaptive Multi-Stage Threshold Accepting Algorithm

5.1 Introduction

In the previous chapter, the performance of existing move acceptance methods based on their characteristics, as classified in the proposed taxonomy from Section 2.3, across multiple instances from a single domain, and across multiple problem domains (cross-domain) was investigated. Two high-level approaches to parameter tuning were compared for configuring the parameters of the move acceptance methods; per-domain where a configuration is found for each domain, and cross-domain where a single configuration is used across all problem domains. The results highlight the shortcomings of the existing designs of move acceptance methods with different move acceptance methods solving some problem domains better than others but do not solve problems from other domains as well as the other move acceptance methods which can outperform them. Moreover, the results showed that the parameter sensitivity of existing move acceptance methods means that when tuned cross-domain, their cross-domain performance is significantly worse than if they had been re-tuned, even at the per-domain level, and this poses a significant obstacle to the design of a cross-domain search method that does not require expert intervention to perform well.

The existing move acceptance methods as discussed in the literature review use a single search strategy. In order for their performance to be improved for solving characteristically different problems, their parameters need to be re-tuned. This is a problem for the design of high-level general-purpose search methods as an expert needs to be involved in the decision-making process for the existing move acceptance methods to achieve a good cross-domain performance. Some of the existing move acceptance methods employ a parameter control method to adapt the behaviour of their strategies used to *adapt* the search trajectory. In other cases, mechanisms are employed to instigate a partial restart, such as in SARH, to explore different regions of the search space. In all cases however, their fundamental search strategies remain largely the same and this limits the ability of existing move acceptance methods to perform well across characteristically different problems. In this chapter, a novel move acceptance method called History-based Adaptive Multi-stage Threshold Accepting (HAMSTA) is proposed which is designed to combine and exploit multiple search strategies to overcome the issues of the existing designs. Moreover, one of these strategies employ a newly designed move acceptance method (HTA) that was designed as a component of HAMSTA which maintains a history of previously accepted solutions such that it is able to *adapt* the search trajectory to that of the local search landscape.

The HAMSTA move acceptance method is explained in Section 5.2. Section 5.3 revisits the experimental design for evaluating its cross-domain performance when compared to the existing move acceptance methods studied in the previous chapter under the local search metaheuristic framework. The results are discussed in Section 5.4, and this chapter is summarised in Section 5.7.

5.2 History-based Adaptive Multi-Stage Threshold Accepting

When used as a move acceptance method for cross-domain search, those investigated in the previous chapter highlighted some key shortcomings of the existing methods. These include:

1. The move acceptance method should have a mechanism to allow it to accept only equal or improving moves if this proves to be beneficial.
2. The move acceptance method should have a mechanism to allow a random walk of the search space accepting a large number of worse cost moves if this proves to benefit the search.
3. Internal parameters that are used to guide the search should be adapted with respect to the problem instance being solved, rather than relying on parameter settings being supplied through instance or domain specific tuning processes. This includes for example any threshold values that should be proportional to actual move delta values.

4. The move acceptance method should not disallow equal cost moves.

History-based Adaptive Multi-Stage Threshold Accepting (HAMSTA) is an iterative multi-stage move acceptance method based on a unique three stage design and is designed to address the aforementioned shortcomings. Each stage uses a different underlying move acceptance method, each either chosen or designed based on a different search strategy. An exploitation stage, IE [53], is used to improve an initial solution, accepting improving or equal moves and rejecting worse quality moves, until the search becomes stuck in a local optimum. An exploration stage, AM [37], is used to allow a number of indiscriminate perturbations to be made to the solution-in-hand, by accepting all moves, with the aim of reaching a solution in a different region of the search landscape. A further stage, HTA — a new move acceptance method as proposed as a component of the HAMSTA move acceptance method and as explained in Section 5.2.5, is used which aims to balance exploitation and exploration based on the search history to find the best solution amongst the current region of possibly multiple local optima. A flow-diagram showing the determination of the termination of each stage and the transitions between them is shown in Figure 5.1. The move acceptance strategies and parameter adaptation procedures are shown in the flow-diagram as grey blocks; these and any associated stage initialisation are detailed in subsections 5.2.2 through 5.2.6. The acceptance of moves under the HAMSTA move acceptance method for each stage are shown in Equation 5.1, assuming $f(.)$ is a minimising objective function, and where τ_i is a threshold value calculated by the history-based threshold accepting algorithm, δ is an estimate of the maximum number of iterations required to encounter a local optimum derived during the IE stage, $f(s_{best_i})$ is the objective value of the best solution found so far, $f(s_{i-\delta+1})$ is the objective value of the solution-in-hand $\delta-1$ iterations previously, equivalent to the objective value of the solution *accepted* δ iterations previously (and this includes those accepted during the AM stage(s)), $f(s_{best_{i-\delta}})$ is the objective value of the best solution that was found up until δ iterations previously.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & \texttt{STAGE == IE} \wedge f(s_i^{'}) \leq f(s_i) \vee \\ & \quad \texttt{STAGE == AM} \vee \\ & \quad \texttt{STAGE == HTA} \wedge f(s_i^{'}) \leq max(f(s_i), \tau_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{5.1}$$

$$where \quad \tau_i = f(s_{best_i}) + f(s_{i-\delta+1}) - f(s_{best_{i-\delta}})$$

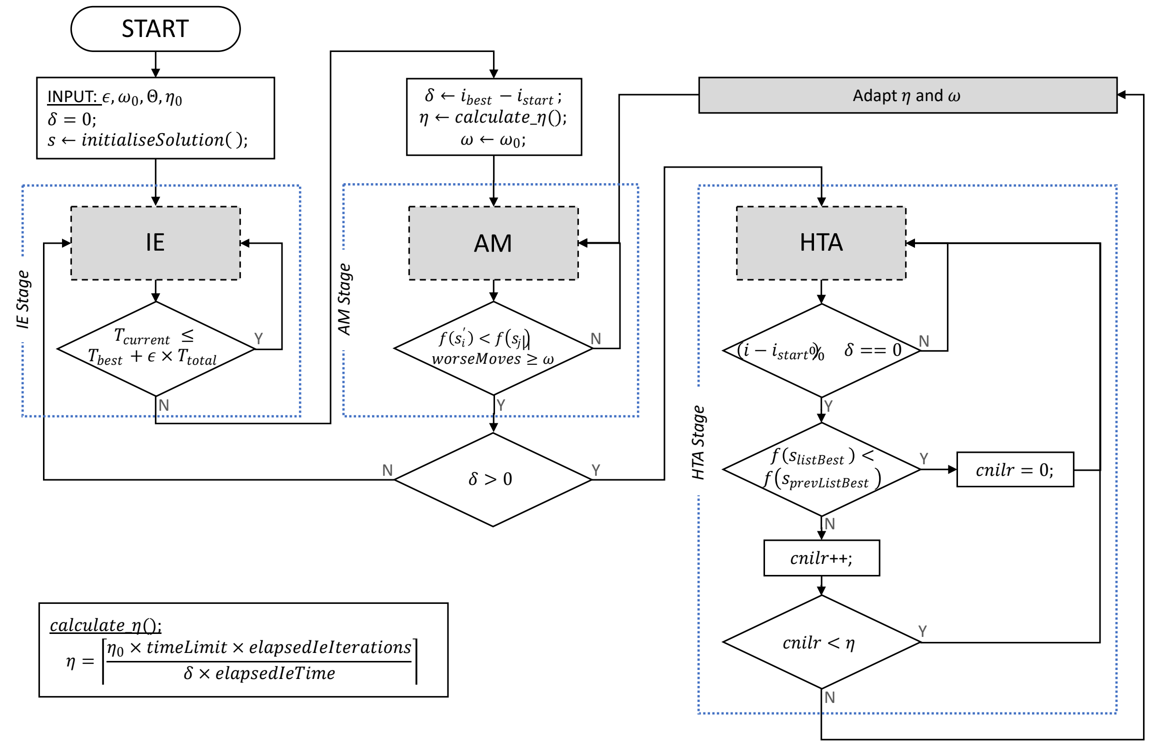

START
INPUT: $\epsilon, \omega_0, \Theta, \eta_0$
$\delta = 0$;
$s \leftarrow initialiseSolution()$;
IE Stage
IE
$T_{current} \leq T_{best} + \epsilon \times T_{total}$
Y
N
$\delta \leftarrow i_{best} - i_{start}$;
$\eta \leftarrow calculate\_\eta()$;
$\omega \leftarrow \omega_0$;
AM Stage
AM
$f(s_i') < f(s_j)$ \| $worseMoves \geq \omega$
N
Y
$\delta > 0$
N
Y
Adapt η and ω
HTA Stage
HTA
$(i - i_{start})\% \quad \delta == 0$
N
Y
$f(s_{listBest}) < f(s_{prevListBest})$
Y
$cnilr = 0$;
N
$cnilr$++;
$cnilr < \eta$
Y
N
calculate_η();
$\eta = \left\lceil \frac{\eta_0 \times timeLimit \times elapsedIeIterations}{\delta \times elapsedIeTime} \right\rceil$

Figure 5.1: Diagram illustrating the transitions between the different stages of HAMSTA, and mechanisms to determine the termination of each stage. Note that there is no predefined end state since the algorithm can terminate during any stage according to the prescribed time-based/iteration-based computational budget. Arrows which pass into stage boundaries include a stage initialisation procedure as defined in their respective stage descriptions. Dashed borders indicate a move acceptance strategy with IE, AM, and HTA stages being discussed in Sections 5.2.2, 5.2.3, and 5.2.4, respectively. Processes filled in grey follow their descriptions from the literature where present (*IE* and *AM*), and *HTA* and the *adaptation of* η *and* ω are explained in Sections 5.2.5 and 5.2.6, respectively. $T_{current}$ and T_{best} are the current time and time that the best solution was found respectively. i represents an iteration count such that i_{best} and i_{start} is the iteration that the best solution was found and the iteration count at the start of the preceding HTA stage.

5.2.1 HAMSTA Parameters

HAMSTA is designed in a way such that its parameters should not need to be re-tuned beyond the design stage. That is, they should not need to be changed in order for HAMSTA to perform well for solving different real-world COPs as the internal control mechanisms all adapt in a meaningful way throughout any given search procedure - this is essential for the design of a cross-domain search method. HAMSTA has four parameters; ϵ, ω_0, ω^+ and η_0, and these are defined as below:

- ϵ - The amount of time (or iterations) as a factor of the overall computational budget that is allocated as a wait time setting to determine the termination of an IE stage.

$$\epsilon \in \{x \mid x \in \mathbb{R},\ 0.0 \leq x \leq 1.0\}$$

- η_0 - The computational budget (time or iterations), as a factor of the overall computational budget, which is used to calculate an initial target number of non-improving list repetitions, η. This in turn is used to determine the termination of an HTA stage, (see Section 5.2.6).

$$\eta_0 \in \{x \mid x \in \mathbb{R},\ 0.0 \leq x \leq 1.0\}$$

- ω_0 - The target number of worse moves ($\delta > 0$) that should be accepted before an AM stage is allowed to terminate. $\omega_0 \in \{x \mid x \in \mathbb{Z},\ x > 0\}$
- ω^+ - An increment value for ω in the event of non-improvement of the best solution during an HTA stage, (see Section 5.2.6). $\omega^+ \in \{x \mid x \in \mathbb{Z},\ x \geq 0\}$

5.2.2 IE Stage

HAMSTA starts with an IE stage which is used initially with two purposes; first to improve the initial solution as best it can without the need for accepting worse moves. This resolves shortcoming #1 as explained at the start of this chapter. Secondly, this stage is able to gain an estimate for the maximum number of iterations of the search that are required to reach a *locally optimal* solution.

As shown in Figure 5.1, there are two transitions into the IE stage; first as the entry point to HAMSTA, and the second from an AM stage. HAMSTA maintains five variables in its memory during an IE stage; T_i, T_{best}, i_{start}, and i_{best}. The variables used in the IE stage and their initial values are as follows:

- $T_{current}$ is the record of the current time (or iteration).
- T_{start} is the elapsed time when the current stage began.
- T_{best} records the time (or iteration) that the best solution was found during the current IE stage, and is initialised as the current time or iteration, $T_{best} \leftarrow T_i$ at the start of each IE stage.
- i_{start} is the iteration count when the most recent IE stage began.
- i_{best} is the iteration count when the best solution was found during the current IE stage.

The pseudo-code for the operation of an IE stage is shown in Algorithm 3. An IE stage iteratively improves a single solution by accepting improving or equal cost moves only, as shown in the *IE Stage* block of Figure 5.1, until an amount of time ($\epsilon \times T_{total}$) has passed without further improving the best solution found during the current IE stage. At the start of an IE stage, the current solution is recorded to allow the best solution found during the current stage to be tracked ($s_{best} \leftarrow s_i$), the current iteration recorded which is used to track the number of iterations taken to find the best solution ($i_{best} \leftarrow i_{start} \leftarrow i$), and the current elapsed time recorded for use by the initialisation of η to estimate the time per iteration ($T_{best} \leftarrow T_{start} \leftarrow T_i$). The IE move acceptance method is then applied (Lines 10 to 14) until an amount of time has passed without further improving the best solution found during the current IE stage (Line 4), and while keeping track of the iteration and time that the best solution was found (Lines 6 to 9). After the IE stage terminates, the number of iterations taken to find the best solution (δ) is recorded for use in the History-based Threshold Accepting algorithm in the HTA stage, and η can be calculated using the elapsed time and iterations during the IE stage. By recording and using δ in subsequent stages, the need for a wait time parameter in the HTA stage(s) is eliminated allowing more time to be spent exploring the search space rather than being stuck in local optima waiting to see if the solution can possibly improve any further.

Algorithm 3: Outline of the IE stage of HAMSTA where; i is the current iteration count, s_i is the current solution-in-hand, ϵ is from the parameter configuration, T_i is the current elapsed time at iteration i, and T_{total} is the total computational budget.

```
   Input: $s_i$, $\epsilon$, $i$, $T_i$, $T_{total}$
1  $s_{best} \leftarrow s_i$;
2  $i_{best} \leftarrow i_{start} \leftarrow i$;
3  $T_{best} \leftarrow T_{start} \leftarrow T_i$;
4  while $T_i \leq T_{best} + \epsilon \times T_{total}$ do
5      $s_i^{'} \leftarrow random \in \mathbb{N}(s_i)$;
6      if $f(s_i^{'}) < f(s_i)$ then
7          $i_{best} \leftarrow i$;
8          $T_{best} \leftarrow T_i$;
9      end
10     if $f(s_i^{'}) \leq f(s_i)$ then
11         $s_{i+1} \leftarrow s_i^{'}$;
12     else
13         $s_{i+1} \leftarrow s_i$;
14     end
15     i++;
16 end
17 $elapsedIeIterations = i - i_{start}$;
18 $elapsedIeTime = T_i - T_{start}$;
19 $\delta = i_{best} - i_{start}$;
20 return $(s_i, \delta, elapsedIeIterations, elapsedIeTime)$;
```

5.2.3 AM Stage

In HAMSTA, an AM stage follows each IE stage and HTA stage which is when the search is thought to be stuck in a *region of optima*. The AM stage is used to perform a number of perturbations to the solution-in-hand by accepting all moves with the aim of finding a solution in a different region of the search landscape. A parameter ω is used to control the number of worse moves ($\delta > 0$) that are accepted in any given AM stage, allowing HAMSTA to either escape a region of local optima with lower settings of ω, or the surrounding region of poor-quality optima with higher settings. The setting of ω is adapted after each HTA stage as described in Section 5.2.6. The idea of counting the number of worse moves rather than total moves in each AM stage is that it is possible to generate moves of equal or improving quality. If the search is stuck on a plateau/neutral region of the search landscape, then it is possible that the target ω is reached without accepting any worse moves and therefore is unable to move to a different region of the search space.

There are two transitions into the AM stage, as shown in Figure 5.1. Upon transition from an IE stage, the settings of η and ω are initialised to their default values as defined by η_0 and ω_0. If the AM stage is transitioned from an HTA stage, then these parameters are set by the parameter adaptation mechanism invoked preceding each HTA stage. HAMSTA maintains three parameters during the AM stage; j, ω, and $worseMoves$. The parameters used in the AM stage, and their initialisation, are as follows:

- j is the iteration that the current stage started and is initialised as the current iteration, $j \leftarrow i$.
- ω is the target number of worse all-moves moves to be accepted and is either initialised to ω_0 if transitioned from an IE stage, otherwise it is left to the parameter adaptation mechanism to control (see Section (5.2.6).
- $worseMoves$ is the current total of worse all-moves moves that have been accepted and is reset to 0 at the start of each AM stage.

Pseudo-code for the operation of an AM stage is shown in Algorithm 4.

5.2.4 HTA Stage

In HAMSTA, the HTA stage follows each AM stage and aims to improve the solution to the best quality solution in the current region of the search space by using search history to balance exploitation and exploration. The HTA stage uses an underlying History-based Threshold Accepting algorithm, as explained in Section 5.2.5, to determine an acceptance threshold (τ) where a move is accepted if the cost of the candidate solution is not worse than the worst of the current solution and

Algorithm 4: Outline of the AM stage of HAMSTA where; s_i is the current solution-in-hand, i is the current iteration count, and ω is the target number of worse all-moves moves to be accepted before terminating the AM stage.

```
   Input: $s_i$, $i$, $\omega$
1  $s_{init} \leftarrow s_i$;
2  $worseMoves = 0$;
3  while $worseMoves < \omega \land f(s_i) \geq f(s_{init})$ do
4      $s_i^{'} \leftarrow random \in \mathbb{N}(s_i)$;
5      if $f(s_i) > f(s_i^{'})$ then
6          $worseMoves$++;
7      end
8      $s_{i+1} \leftarrow s_i^{'}$;
9  end
10 return $s_i$;
```

a threshold value, τ. The acceptance of moves under the HTA algorithm is illustrated in Equation 5.2 and incorporated into the above HAMSTA acceptance in Equation 5.1.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq max(f(s_i), \tau_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{5.2}$$

HAMSTA maintains in memory during an HTA stage the parameters δ, and η, from which δ is derived from the IE stage, and η is controlled by the parameter adaptation mechanism of HAMSTA. In addition, HAMSTA also maintains the variables; i_{start}, *cnilr*, $f(s_{listBest})$, and $f(s_{prevListBest})$. The parameters and variables used in the HTA stage, and their initialisation, are as follows:

- δ is the record of the estimated maximum number of iterations required to find a local optimum, as derived from the IE stage as the number of iterations taken to find the best solution and determines the length of the threshold history list.

- η is the target number of consecutive non-improving repetitions of the threshold history list and is controlled by the parameter adaptation mechanism of HAMSTA. When *cnilr* $== \eta$, the HTA stage terminates.

- i_{start} keeps a record of the iteration count when the HTA stage began and is initialised as, $i_{start} \leftarrow i$. It is used by HTA to determine when a full repetition of the threshold history list has been achieved.

- *cnilr* maintains a record of the number of consecutive non-improving repetitions of the threshold history list, initialised as *cnilr* $\leftarrow 0$, and is used by HAMSTA to determine when the HTA stage should end.

- $f(s_{listBest})$ keeps track of the objective value of the best solution found during the current repetition of the threshold history list within the HTA algorithm and is re-initialised at the start of each list repetition as $f(s_{listBest}) \leftarrow f(s_i)$.
- $f(s_{prevListBest})$ keeps track of the objective value of the best solution found during the previous repetition of the threshold history list within the HTA algorithm and is re-initialised at the start of each list repetition as $f(s_{prevListBest}) \leftarrow f(s_{listBest})$.
- The roles of $f(s_{listBest})$ and $f(s_{prevListBest})$ are to allow the HTA algorithm to detect whether the current repetition of the threshold history list was improving or non-improving, and hence supports the *cnilr* parameter.

The operation of a HTA stage and HTA algorithm is as follows, and as shown using pseudo-code in Algorithm 5. A random perturbation is made to the solution-in-hand to produce a neighbouring solution, $s_i^{'}$. A threshold τ_i is calculated according to the HTA algorithm as $\tau = f(s_{best_i}) + f(s_{i-\delta+1}) - f(s_{best_{i-\delta}})$ from which the move is either accepted ($f(s_i^{'}) \leq \max(f(s_i), \tau)$) or rejected ($f(s_i^{'}) > \max(f(s_i), \tau)$) where; $f(s_{best_i})$ is the objective value of the best solution found so far, $f(s_{i-\delta+1})$ is the objective value of the current solution accepted $\delta - 1$ iterations previous (the objective value of the accepted solution δ iteration previous), and $f(s_{best_{i-\delta}})$ is the objective value of the best solution δ iterations previous. That is, the threshold value tracks with the objective value of the best solution to promote higher exploitation effects as the best solution continues to be improved. If the best solution has been improved, then it is updated as the candidate solution, and the *bestFound* flag updated. At this step, the best solution found during the current list repetition is also updated. This process repeats until a number of iterations has passed which is a multiple of δ (a full list repetition has been performed), at which point the *maxCnilrToBest* is updated with the help of two other tracking variables $f(s_{listBest})$ and $f(s_{stageBest})$, and the current value of *cnilr* is updated depending on whether the current list repetition found a solution which improved the best solution found so far during the current HTA stage (*cnilr* $= 0$), or not (*cnilr*++). The HTA stage continues until a consecutive number of non-improving list repetitions (*cnilr*) has elapsed that exceeds an upper limit (η).

5.2.5 History-based Threshold Accepting

The history-based threshold accepting algorithm (HTA algorithm) is an *adaptive non-stochastic threshold* move acceptance method that is synonymous to that of Late Acceptance (LA) [28], but

Algorithm 5: Outline of the HTA *stage* of HAMSTA where; s_i is the current solution-in-hand, i is the current iteration count, δ is the estimated maximum number of iterations required to encounter a local optimum (as derived from the IE stage), and η is the maximum number of consecutive non-improving list repetitions permitted before the HTA stage should terminate.

```
Input: $s_i$, $i$, $\delta$, $\eta$
$i_{start} \leftarrow i$;
$cnilr = 0$;
$maxCnilrToBest = 0$;
$bestFound = false$;
$s_{listBest} \leftarrow s_i$;
$s_{prevListBest} \leftarrow s_i$;
$s_{stageBest} \leftarrow s_i$;
while $cnilr < \eta$ do
    $s_i^{'} \leftarrow random \in \mathbb{N}(s_i)$;
    $\tau = f(s_{best_i}) + f(s_{i-\delta+1}) - f(s_{best_{i-\delta}})$;
    if $f(s_i^{'}) \leq \max(f(s_i), \tau)$ then                         // Accept
        $s_{i+1} \leftarrow s_i^{'}$;
    else                                                                // Reject
        $s_{i+1} \leftarrow s_i$;
    end
    if $f(s_i^{'}) < f(s_{best})$ then                                  // Update best
        $s_{best} \leftarrow s_i^{'}$;
        $bestFound = true$;
    end
    if $f(s_i^{'}) < f(s_{listBest})$ then                              // Update list best
        $s_{listBest} \leftarrow s_i^{'}$
    end
    if $(i - i_{start}) \mod \delta == 0$ then // Adapt cnilr after each full list repetition
        if $f(s_{listBest}) < f(s_{stageBest})$ then
            $maxCnilrToBest = \max(cnilr, maxCnilrToBest)$;
            $s_{stageBest} \leftarrow s_{listBest}$;
            $cnilr = 0$;
        else
            $cnilr$++;
        end
        $s_{prevListBest} \leftarrow s_{listBest}$;
        $s_{listBest} \leftarrow s_{i+1}$;
    end
    $i$++;
end
return $(s_i, bestFound, maxCnilrToBest)$;
```

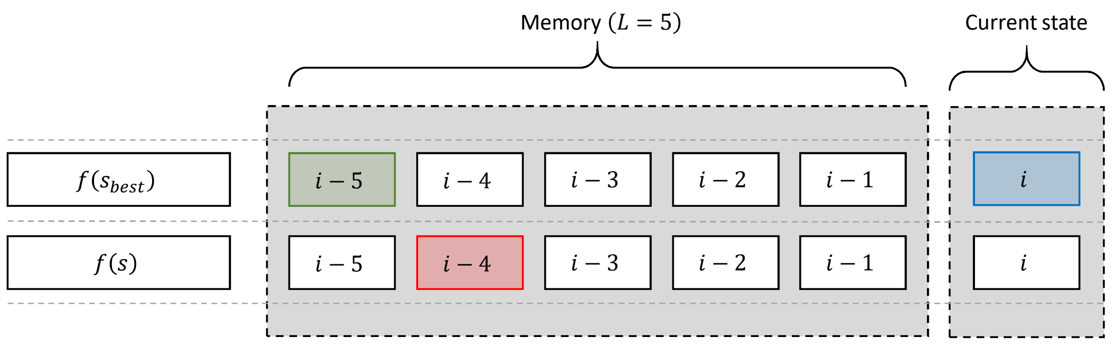
with the addition of a mechanism that promotes higher exploitation effects whenever the best solution is improved. This stage is designed in such a way that shortcomings #3 and #4 are addressed. That is, that threshold values are proportional to the problem instance being solved, as they are learnt during the search, and that no equal cost moves are disallowed by restricting acceptance to improving moves only. Furthermore, due to the combination of HTA as described below, and the adaptation of the number of list repetitions with respect to how well the search is progressing, shortcoming #2 is also addressed. In this section, the HTA algorithm is outlined, and various list initialisation strategies are explored — illustrating the performance and range of behaviours that can be achieved by HTA, including that that is ultimately used when used as the move acceptance method for the exploration/exploitation stage within HAMSTA.

HTA Definition

The acceptance mechanism of the HTA algorithm is shown in Equation 5.3. Like all non-stochastic threshold move acceptance methods from the literature, a move is accepted if the objective value of the candidate solution is not worse than the maximum (worst) of the current solution, and a threshold value. That is, $f(s_i^{'}) \leq \max(f(s_i), \tau_i)$.

$$s_{i+1} \leftarrow \begin{cases} s_i^{'} & f(s_i^{'}) \leq max(f(s_i), \tau_i) \\ s_i & \texttt{otherwise} \end{cases} \tag{5.3}$$
$$\texttt{where } \tau_i = f(s_{i-L+1}) - f(s_{best_{i-L}}) + f(s_{best_i})$$

The HTA algorithm combines strategies used by Late Acceptance (LA) and Record-to-record Travel (RRT) to exploit the features of characteristically different search spaces using only a single algorithm whereby; a list of fixed length containing the objective values of the L previously accepted solutions is maintained for calculating a threshold value based on the objective value of the solution accepted a number of iterations prior to the current iteration (as used by LA). The idea behind maintaining a memory of solution values is that the HTA algorithm can, when combined with a suitable list length, "learn" the landscape feature(s) present in the problem being solved which can then aid a more effective search process. Moreover, a secondary list of the same nature is maintained, but contains the objective values of the best solution found at the given iteration. This allows HTA to reduce the threshold values over time whenever the best solution is improved (as used by RRT). The idea behind employing this strategy is that if a search procedure is in an area of the search landscape region where the best solution is being improved, then exploitation should be favoured over exploration. That is, the number of worse moves accepted should be decreased in time and irrespective of the move operator employed. Thus, one of the advantages that HTA has over LA is that the threshold values used by HTA have the ability to change as the best solution is improved (adapt to the current search state); hence, the threshold values are decreased proportional to any improvement in the best solution found so far and promoting stronger exploitation effects when compared to the standard LA. If the search does not enforce the solution to be improved over



$$\tau_i = f(s_{i-L+1}) - f(s_{bes\ _{i-L}}) + f(s_{best_i})$$

where

$f(s_{i-L+1})$ is the solution accepted L iterations previously.

$f(s_{best_{i-L}})$ is the best solution found as of L iterations previously.

$f(s_{best_i})$ is the current best solution.

Figure 5.2: Illustration of the calculation of the threshold value (τ_i) according to the History-based Threshold Accepting algorithm (HTA) using a memory of objective function values of the best solutions ($f(s_{best})$) and current solutions ($f(s)$) from the current iteration, i, to those L iterations previous, $i - L$.

time, such as with TA and NA as explored in the previous chapter, then problems which have move operators that generate a high number of worse candidate moves are unlikely to perform well. This method of exploitation exploits the chance that if we are in an area of the search landscape region where the best solution can be improved, then we should favour exploitation over exploration. If the search is in a region of the search space where the best solution is not being improved, then this exploitation is naturally scaled back, since the objective value of the best solution is not improving, and allows the search to escape from some regions of local optima which it would otherwise become trapped, and is then equivalent to LA.

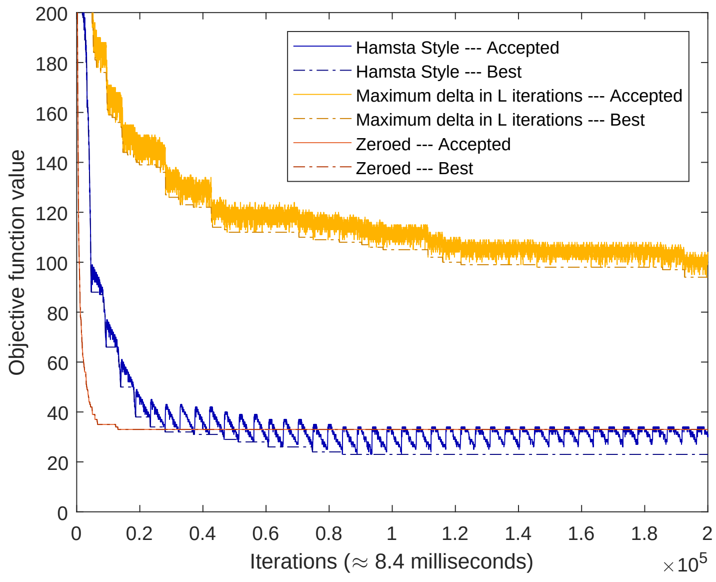
In the HTA algorithm, the threshold value τ_i is calculated at iteration i based on a fixed length (L) memory of the objective values of; the accepted solution a number of iterations previous ($f(s_{i-L+1})$), the best solution found as of the same number of iterations previous ($f(s_{best_{i-L}})$), and the current best solution found ($f(s_{best_i})$). The calculation of the threshold value is illustrated in Figure 5.2 and highlights the memory indices (with respect to the current iteration) that are used for both the histories of the best and current solutions.

List Initialisation Strategies

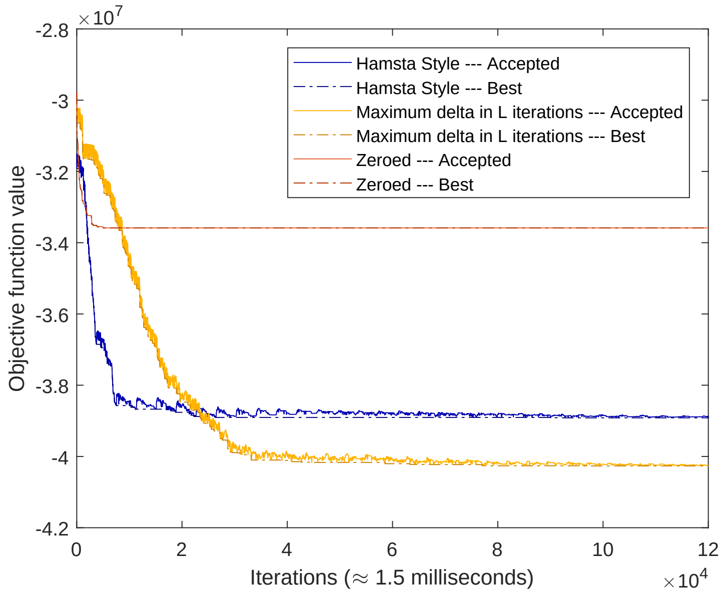
There are two design choices to be made for the design of the HTA algorithm. First is the length of the list, and the second is the list initialisation strategy. As HTA is synonymous to LA, the effects of the setting of the list length parameter is the same as in LA where a small (large) list length provides a fast (slow) convergence time, but a relatively poor (good) quality solution as a result [177]. Since HTA is designed for use in HAMSTA to provide a well performing search strategy by exploiting the search space features, the list length is set equal to the number of iterations taken to encounter the local optimum from the initial solution. That is, an *estimate* for the maximum number of iterations required to encounter a local optima. The second design choice is the list initialisation strategy, and multiple strategies are compared below.

Three strategies are explored for the list initialisation strategy. The first two are simple strategies where all elements of the list are set to the same value; first, the maximum move delta (Δ_{max}) over the first L iterations is used, and the second is to set all values to zero, and this simulates IE move acceptance. An infinite strategy was also explored but preliminary analysis showed this to perform significantly worse than the strategies explored herein; hence, traces using such strategy are omitted for brevity. The third strategy that is explored follows the style of the values that eventually populate the HTA list under the HAMSTA move acceptance method due to the use of the All-Moves stage, and this causes the values in the list to decrease over the length of the list. For the sake of comparison, this is emulated by initialising the values in the list such that they linearly decrease from an initial value equal to Δ_{max} (as previous) to zero; however, in practise, these values will follow the terrain of the local search landscape region.

Figures 5.3a, 5.3b, and 5.3c show the accepted and best solution values for the three list initialisation strategies for solving an instance of MAX-SAT, Max Cut, and Quadratic Assignment problems respectively. The zero-ing strategy, equivalent to the IE move acceptance method, has a superior convergence rate when compared to the HAMSTA-style and Δ_{max} strategies; however, always obtains inferior solutions. The HAMSTA-style and Δ_{max} list initialisation strategies outperform the zero-ing (IE) strategy, except for the case of Δ_{max} when solving the MAX-SAT problem instance. In all cases, the HAMSTA-style strategy can find high quality solutions faster than Δ_{max}, but Δ_{max} can eventually outperform the HAMSTA-style strategy at larger computational budgets. This however could be due in part to the HAMSTA-style list initialisation using emulated values rather than ones that have been "learnt" from the search space of the problem that it is solving itself; as mentioned before, such investigation is however left for future studies.

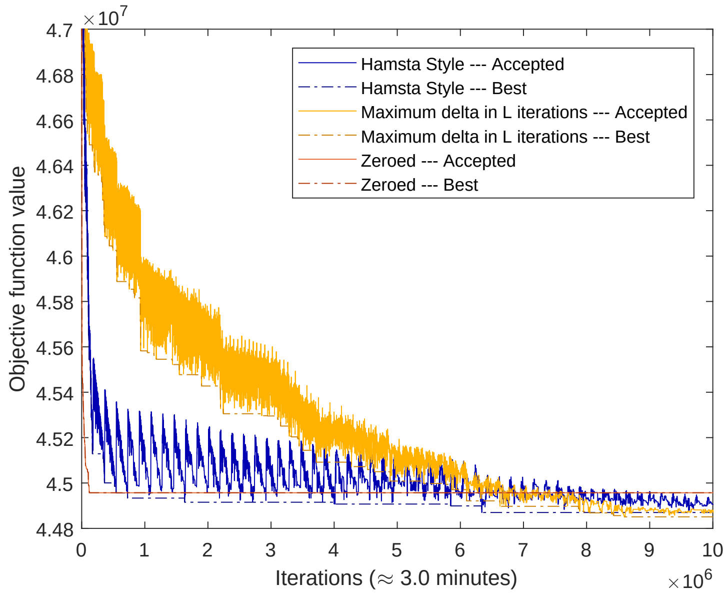


(a) HTA solving instance ID #3 of the MAX-SAT problem. While zero-ing (equivalent to IE) is superior to initialising the list using Δ_{max}, a HAMSTA-style initialisation proves superior to the zero-ing strategy in this case.



(b) HTA solving instance ID #0 of the Max Cut problem. Both Δ_{max} and a HAMSTA-style initialisation perform superior to the zero-ing strategy with the former finding a better quality solution, but in a longer time frame.

Figure 5.3: Accepted and best solution values for the HTA algorithm solving; (a) instance ID #3 of the MAX-SAT problem, (b) instance ID #0 of the Max Cut problem, and (c) instance ID #7 of the Quadratic Assignment problem with three different list initialisation procedures.



(c) HTA solving instance ID #7 of the Quadratic Assignment problem. Both Δ_{max} and a HAMSTA-style initialisation perform superior to the zero-ing strategy. While both the HAMSTA-style and Δ_{max} strategies perform similarly, the HAMSTA-style strategy is able to find a high quality solution much faster than the latter, and at a rate not too dissimilar from the zero-ing strategy.

Figure 5.3: Accepted and best solution values for the HTA algorithm solving; (a) instance ID #3 of the MAX-SAT problem, (b) instance ID #0 of the Max Cut problem, and (c) instance ID #7 of the Quadratic Assignment problem with three different list initialisation procedures.

Convergence plots of the HTA algorithm using a HAMSTA-style list initialisation strategy using objective values of the accepted solutions are shown in Figure 5.4. Figures 5.4a, 5.4c, and 5.4e illustrate the convergence of HTA for solving an instance of the SAT, MAC, and QAP problems respectively over a 3-minute computational budget. Figures 5.4b and 5.4d are focused versions of 5.4a and 5.4c respectively, focusing on the time period to convergence. All plots indicate that HTA runs to convergence, but presents a peculiar case when solving an instance of the SAT problem where a high-quality solution is found early-on, but the search diversifies slightly before converging on a sub-optimal solution. When solving MAC instance #0, HAMSTA finds a high quality solution early on which is not able to be improved.

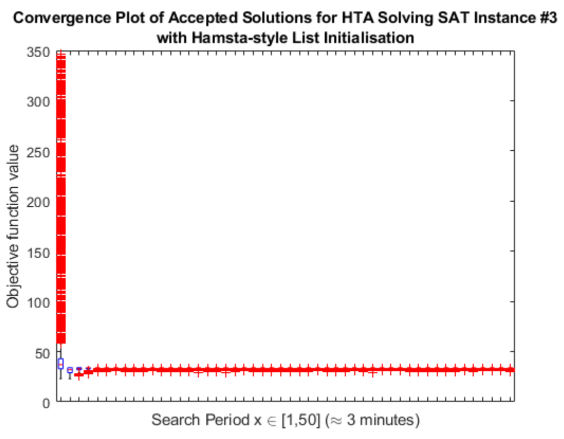
Figure 5.5 compares the progress plots of HTA with HAMSTA-style list initialisation to its derivatives, Late Acceptance (LA) and Record-to-record Travel (RRT), as well as IE for completeness. The list lengths for HTA and LA are set equally to the number of iterations required to find the best solution using an IE search strategy. The threshold value for RRT, and the starting value for the HAMSTA-style list initialisation of HTA, is set as Δ_{max}. The progress plots illustrate that the novel hybridisation of LA and RRT to form HTA has a superior performance compared to its derivatives in this case.

5.2.6 Parameter Adaptation

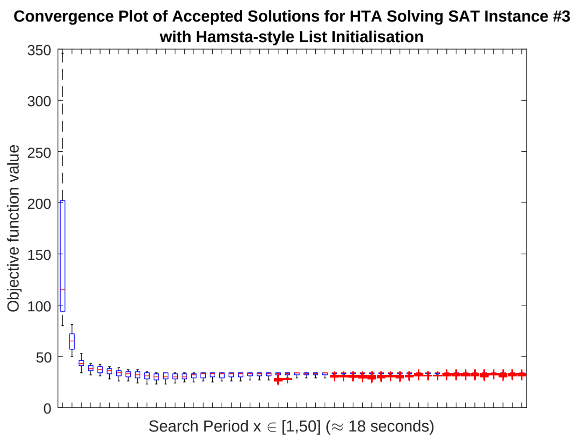
The ability of HAMSTA to search across characteristically different search spaces is aided by the adaptation of its internal parameters η and ω. After each HTA stage has been completed, these parameter settings are adapted based on the search history to optimise the internal mechanisms that guide the search process by controlling the *magnitude* of the exploration (ω), and minimising, where appropriate, the exploration budget for any given HTA stage (η).

Initial setting of η

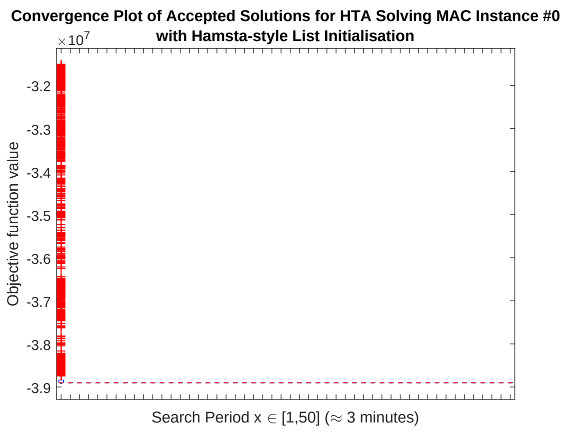
The first time that the HTA stage is used, η is set according to η_0, and δ from the IE stage, and is equal to the number of list repetitions (rounded up) that can be achieved in a predefined computational budget of the total search budget. If HAMSTA is using an iteration-based termination criterion, then η can be calculated as shown in Equation 5.4. Otherwise, if HAMSTA is using a time-based termination criterion, then the elapsed time from the most recent IE stage and number of iterations executed in the most recent IE stage are used to estimate the time taken for each iteration such that η can be calculated as shown in Equation (5.5).



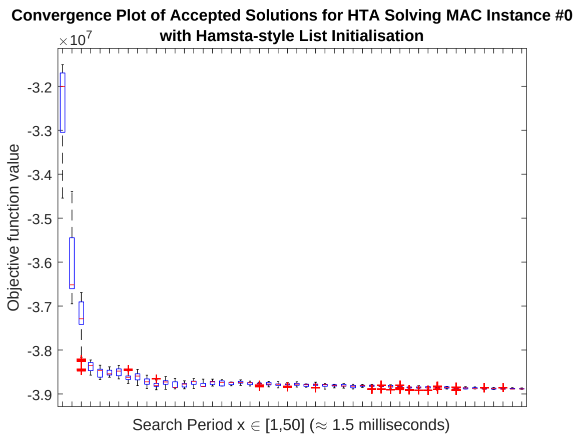
(a)



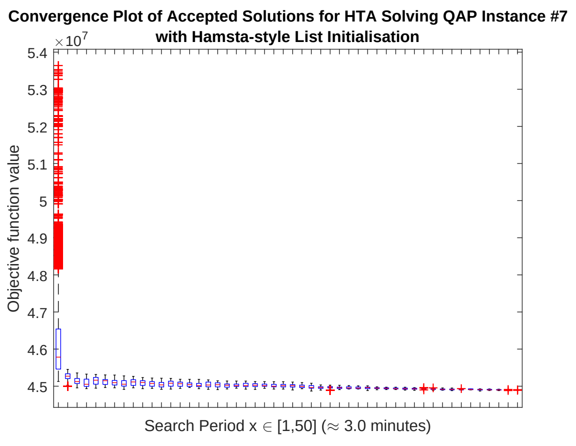
(b)



(c)



(d)



(e)

Figure 5.4: Boxplots of the objective values of accepted solutions illustrating the convergence of HTA over three characteristically different problem domains. '+' marks symbolise statistical outliers according to either $f(s) > q_3 + 1.5 \times (q_3 - q_1)$ or $f(s) < q_1 - 1.5 \times (q_3 - q_1)$ where q_1 and q_3 are the 25th and 75th sample data percentiles.

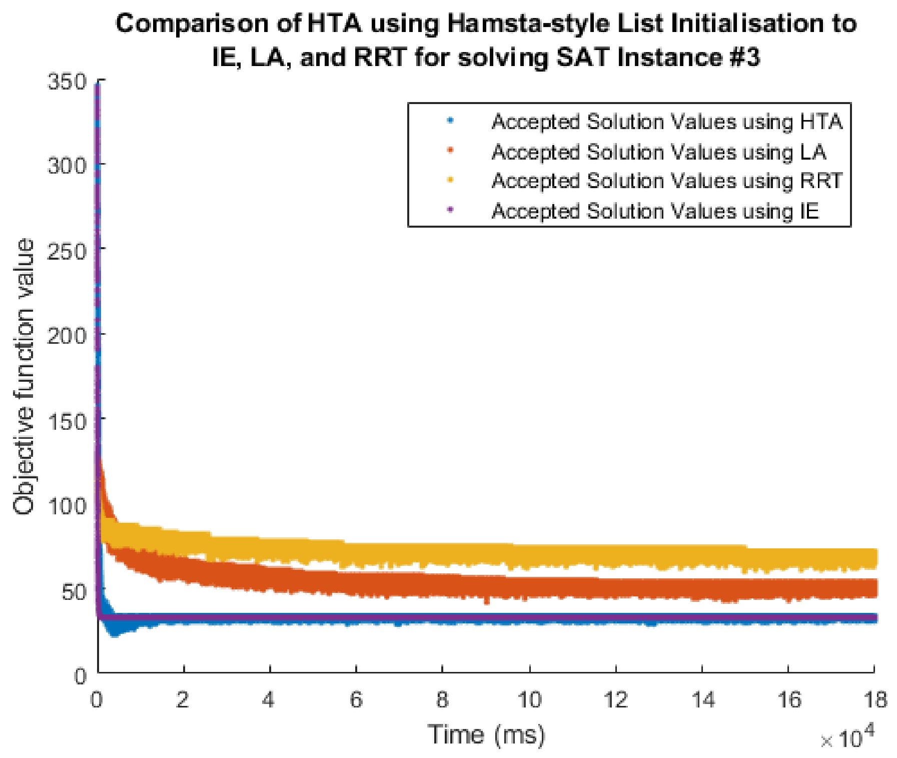


Figure 5.5: Comparison of the accepted solution values for the HTA algorithm using a HAMSTA-style list initialisation strategy for solving instance ID #3 of the MAX-SAT problem compared to IE, LA, and RRT. The list length for LA is equal to that used in HTA which is equal to Δ_{max}. Both HTA and IE have a faster convergence than LA and RRT with HTA outperforming IE, and while also finding better quality solutions given the 3 minute computational budget.

$$\eta = \left\lceil \frac{\eta_0 \times iterationLimit}{\delta} \right\rceil \tag{5.4}$$

$$\eta = \left\lceil \frac{\eta_0 \times timeLimit \times elapsedIeIterations}{\delta \times elapsedIeTime} \right\rceil \tag{5.5}$$

Adaptation of η

The purpose of η is to control the number of consecutive non-improving list repetitions before terminating a HTA stage. If this setting is too high, then a high proportion of the computational budget is wasted searching in the same local optimum; however, if this setting is too low, then it is possible that the HTA stage is prematurely ended meaning that better quality solutions are left undiscovered. It is therefore desirable to start with a high-enough setting for η such that high quality solutions are not missed in most search spaces, but to control η such that its setting converges on a value over time that is better suited for the search space of the problem being solved.

The parameter adaptation mechanism requires three parameters which are used to determine the setting for η; η, η_{best}, *bestFoundInCurrentStage*, and η_{lb} where η is the current setting, η_{best} is the setting of η when the best solution was found, *bestFoundInCurrentStage* shows whether the best solution found was found in the current HTA stage, and η_{lb} is the maximum number of consecutive non-improving list repetitions that were required to find the best solution in the current HTA stage - an estimate for the lower bound for η. η is adapted based on the scheme shown in Equation (5.6).

$$\eta \leftarrow \begin{cases} \left\lceil \dfrac{\eta + \eta_{lb}}{2} \right\rceil & bestFoundInCurrentStage \\ \eta_{best} & \texttt{otherwise} \end{cases} \tag{5.6}$$

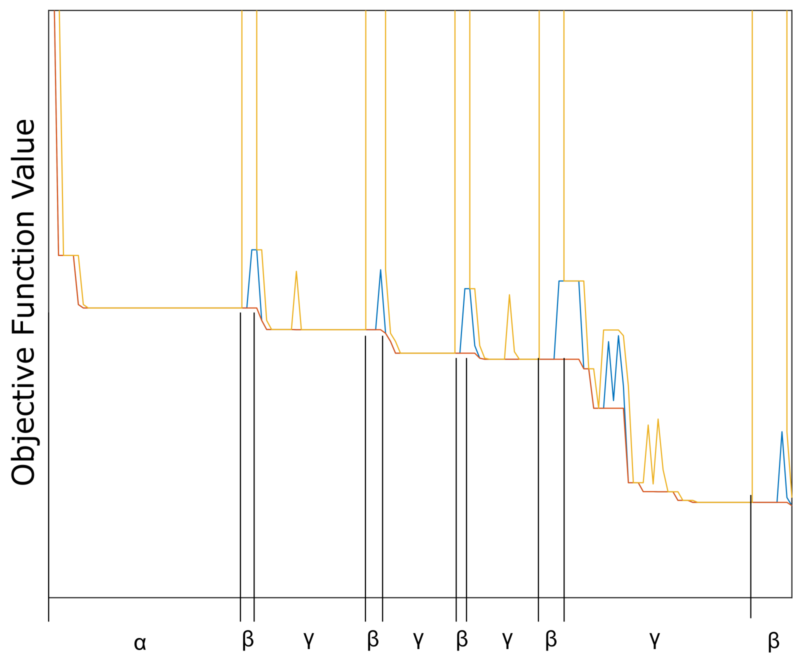


Figure 5.6: Annotation of the stages in an execution of the HAMSTA algorithm where; α is the initial IE stage (Lines 4-8 of Algorithm 2), β is an AM_HTA stage (Lines 10-15 of Algorithm 2), and γ is an HTA stage (Lines 19-24 of Algorithm 2). Note that the yellow trace represents the threshold value, which is infinite during an AM stage, the blue trace represents the objective value of the current solution, and the orange trace represents the objective value of the best solution found so far.

Adaptation of ω

The purpose of ω is to control the number of worse all-moves moves that are accepted in each AM stage and hence decide the termination of each AM stage. When HAMSTA becomes stuck in a local optimum or region of optima, and the HTA stage is unable to escape it/them, it is then

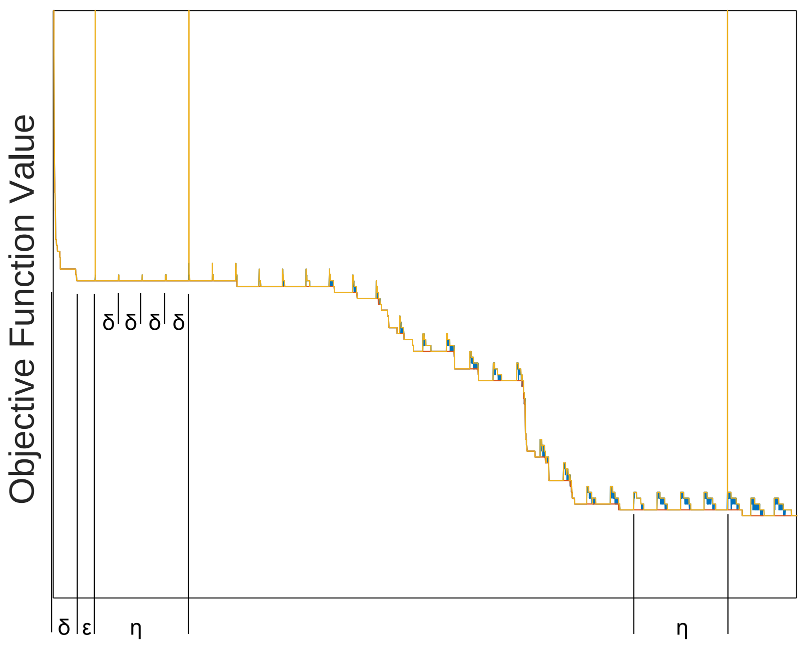


Figure 5.7: Annotation of the algorithmic parameters of an actual execution of the HAMSTA algorithm where; δ is the number of iterations to find the best solution in the IE stage, ϵ is the amount of time that should pass after finding the best solution in the IE stage before entering the AM stage, and η is the maximum consecutive non-improving list repetitions value dependant on the η_0 parameter and internal adaptation mechanisms. Note that the yellow trace represents the threshold value, which is "infinite" during an AM stage, the blue trace represents the objective value of the current solution, and the orange trace represents the objective value of the best solution found so far.

necessary to accept some worse moves. In the HAMSTA algorithm, the AM stage is used to escape such regions of local optima. By accepting a higher number of worse moves, the search can reach different regions of the search space. If HAMSTA, however, is in a promising region of the search space, then it is favourable to accept only a few worse moves in an attempt to find a better quality region of the search space close to that of the solution-in-hand. By adapting ω, the aim is to control this exploration phase of the search by exploiting the history from the HTA stage.

The parameter adaptation mechanism requires three parameters which are used to determine the setting for ω; ω, ω^+, and a boolean *bestFoundInCurrentStage* which shows whether the current HTA stage was able to improve the best solution found. ω is adapted based on the scheme shown in Equation (5.7).

$$\omega \leftarrow \begin{cases} \omega_0 & bestFoundInCurrentStage \\ \omega + \omega^+ & \texttt{otherwise} \end{cases} \tag{5.7}$$

5.3 Experimentation

The aim of the work in this chapter is to contribute a novel move acceptance method which, as a component of a local search metaheuristic, has a cross-domain performance that is as-good-as the existing move acceptance methods that are re-tuned for each problem domain, *but without itself requiring re-tuning beyond its design stage.* That is to say HAMSTA uses a single cross-domain parameter configuration. The cross-domain performance of HAMSTA, as the proposed method, tuned only cross-domain is therefore compared to the eight other move acceptance methods that were investigated in the previous chapter, and that are themselves tuned both cross-domain (π_{xd}), and per-domain (π_d) where plausible, across the full set of 45 problem instances spanning 9 problem domains as used previously under a single-point based local search metaheuristic framework. The experimental framework and move acceptance method configurations follow those as detailed in the methodology given in Section 3. The parameter configuration used for HAMSTA is given in Section 5.3.1 along with the results of the respective parameter tuning experiments.

In cohesion with the work in the previous chapter, the cross-domain performance of the move acceptance methods are compared using their μ_{norm} scores. A non-parametric variant of the one-way ANOVA with repeated measures test in the form of the Friedman's test is also used to compare the statistical significance between the performance of each of the move acceptance methods.

In addition to boxplots of the $f_{norm}(s)$ scores being used to illustrate the performance of each move acceptance method, an experimental cumulative distribution function (ECDF) plot is proposed as a more illustratively descriptive statistic of the comparative cross-domain performance of multiple algorithms. The ECDF plot uses the $f_{norm}(s)$ scores from each trial to form a cumulative distribution of them, and such distribution is plotted for each move acceptance method. The area under the ECDF shows the cross-domain performance of a given move acceptance method, m, and this area is inversely proportional to the the respective μ_{norm} score such that $\int_0^1 ecdf(m)dm = 1 - \dfrac{\mu_{norm}(m)}{n}$ where n is the total number of trials.

5.3.1 Cross-Domain Parameter Tuning

As with previous move acceptance methods that were tuned cross-domain, HAMSTA is tuned on a subset of domains and instances. This means that the remaining 5 domains are completely unseen and hence HAMSTA is evaluated without any additional expert intervention. HAMSTA, containing four parameters, is tuned for cross-domain search using the Taguchi Design of Experiments method as used previously for tuning move acceptance methods with three or more parameters. The parameters of HAMSTA were tuned using a 4-factor 5-level L25 Orthogonal Array design where; $\epsilon \in \{0.01, 0.02, 0.05, 0.10, 0.20\}$, $\omega_0 \in \{0.01, 0.02, 0.05, 0.10, 0.20\}$, $\omega^+ \in \{1, 2, 3, 4, 5\}$, and $\eta_0 \in \{0, 1, 2, 3, 4\}$.

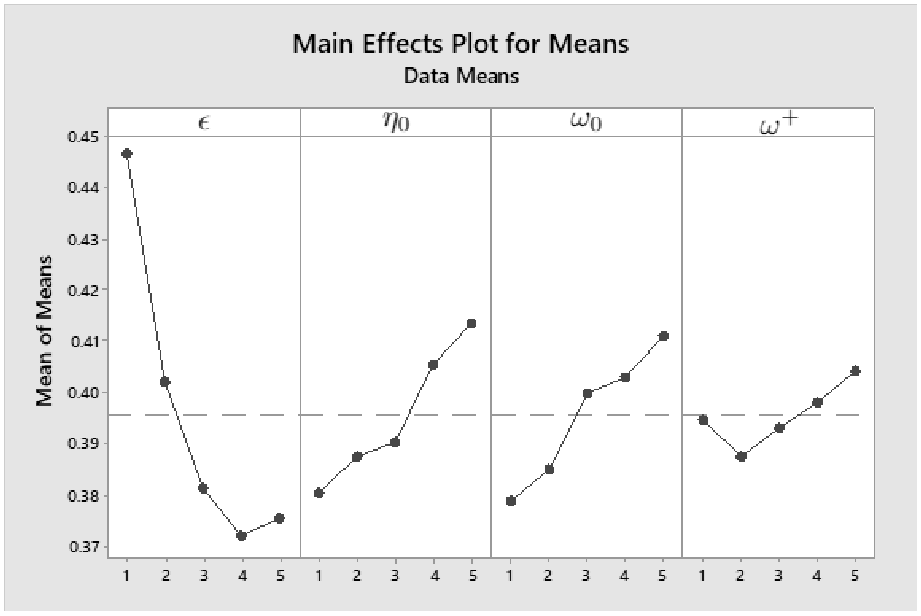


Figure 5.8: Results of tuning the parameters of HAMSTA for cross-domain search using the Taguchi DOE with a 4-factor 5-level L25 Orthogonal Array. Reported means are mean μ_{norm} values from the 8 training instances computed over all 25 parameter configuration experiments. Parameter values at each level for each parameter are as follows; $\epsilon\,[\{1, 2, 3, 4, 5\} \mapsto \{0.01, 0.02, 0.05, 0.10, 0.20\}]$, $\eta_0\,[\{1, 2, 3, 4, 5\} \mapsto \{0.01, 0.02, 0.05, 0.10, 0.20\}]$, $\omega_0\,[\{1, 2, 3, 4, 5\} \mapsto \{1, 2, 3, 4, 5\}]$, and $\omega^+\,[\{1, 2, 3, 4, 5\} \mapsto \{0, 1, 2, 3, 4\}]$.

The results of performing tuning show that the best settings to use for cross-domain search is $\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \omega_0 = 0.01, \omega^+ = 1, \eta_0 = 1\}$. The results also highlight the importance of having a sufficiently large waiting time (ϵ) with smaller settings having a detrimental affect on HAMSTA's cross-domain performance. When used as a search method for cross-domain search, HAMSTA benefits from small settings of η_0, ω_0, and ω^+. η_0, affecting the permissible number of consecutive non-improving list repetitions before an HTA stage is terminated, has the best setting equal to the smallest tested (0.01) meaning that if the time since the best solution found during the current HTA stage exceeds 1% of the total computational budget, then the HTA stage is terminated. The general trend apparent from the results in Figure 5.8 for improved performance at smaller settings would also suggest that further improvement in the cross-domain performance of HAMSTA may be possible if further tuning is conducted at even lower settings. The initial number of worse all-moves moves (ω_0) is required to be at least 1 in order to both utilise an AM stage, and populate the list of move deltas within the HTA algorithm. ω_0 was therefore tuned with the

Table 5.1: Parameter configuration of HAMSTA tuned per-domain across each of the nine benchmark problem domains using 1 small and 1 large selected instance for tuning of each domain.

Domain	ϵ	η_0	ω_0	ω^+
BP	0.20	0.01	1	1
FS	0.20	0.20	1	1
PS	0.02	0.02	2	2
SAT	0.01	0.05	1	1
TSP	0.20	0.20	1	1
VRPTW	0.01	0.10	4	3
KP	0.02	0.01	1	0
MAC	0.01	0.10	5	2
QAP	0.01	0.10	1	0

values $\omega_0 \in \{1, 2, 3, 4, 5\}$ and the best setting for cross-domain search was equal to 1. ω^+ on the other hand determines the value by which ω is incremented, as decided by one of the parameter adaptation mechanisms of HAMSTA. For tuning, the possibility to disable this mechanism was also explored, essentially causing ω to be fixed for the duration of the search, and was hence tuned with the settings $\omega^+ \in \{0, 1, 2, 3, 4\}$. The results of tuning ω^+ showed that disabling this mechanism ($\omega^+ = 0$) caused the cross-domain performance of HAMSTA to deteriorate with the best setting being $\omega^+ = 1$ when tuned for cross-domain search.

To allow the behaviours of HAMSTA to be compared across solving characteristically different COPs in the form of progress plot analysis, additional tuning was performed at a per-domain level to gain insights into its behaviour; the results of which are summarised in Table 5.1.

The presence of different parameter settings for different problem domains shows that while HAMSTA is designed for cross-domain search, inevitably and unsurprisingly its performance is still sensitive to its parameter values. With the exception of the Personnel Scheduling and 0-1 Knapsack problem domains, some domains require the smallest setting for ϵ to achieve the best performance whereas for others, the largest setting was favoured. Looking back at the results from Section 4.2, the domains that preferred the highest setting for ϵ correlate to those domains that were best solved by the IE move acceptance method, but there is no other obvious correlation with respect to the lower settings of ϵ. Comparison of the progress plots for such domains, irrespective of the parameter tuning approach, highlights the differences in their search landscapes and hence the requirement for such parameter settings in each case. The settings for η_0 do not seem to follow any trends however a higher setting can be seen to aid those domains where few worse moves are required to achieve best performance. The settings for ω_0 and ω^+ appear to be inter-linked with low settings of ω_0 corresponding to low settings of ω^+ and vice-versa for higher settings. In 2 out of the 9 domains, KP and QAP, disabling the strategy of incrementing ω proved beneficial to the performance of

HAMSTA for solving those domains. Comparison of the progress plots for these domains shows a significant change in behaviour of HAMSTA going from a seeming random-walk when using the cross-domain parameter configuration $\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 1\}$ to a more reasonable search trajectory using their per-domain parameter configurations, $\pi_d^{\text{HAMSTA}}(\text{KP}) = \{\epsilon = 0.02, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 0\}$ and $\pi_d^{\text{HAMSTA}}(\text{QAP}) = \{\epsilon = 0.01, \eta_0 = 0.10, \omega_0 = 1, \omega^+ = 0\}$ for KP and QAP respectively.

5.4 Experimental Results

The results comparing HAMSTA to the existing move acceptance methods are shown in Table 5.2. The top portion of the table shows the results for comparing HAMSTA ($\pi_{\texttt{xd}}$) to the existing move acceptance methods that are also tuned for cross-domain search. The bottom portion of the table shows the results for comparing HAMSTA ($\pi_{\texttt{xd}}$) to the existing move acceptance methods that can be re-tuned for each problem domain.

The cross-domain tuned comparison shows that HAMSTA has the best cross-domain performance with a μ_{norm} score of 254.19, and with the 2nd best move acceptance method being SA with a μ_{norm} score of 360.82. The results of performing a Friedman's test shows that there is a significant performance difference between the tested cross-domain tuned move acceptance methods $\big(p = 0.00, \chi^2(8) = 3232.93\big)$. Post-hoc analysis shows that while HAMSTA has the best cross-domain performance, it does not statistically significantly differ from SA, IE, or AILLA with μ_{norm} scores of 360.82, 416.61, and 500.64 respectively. The per-domain tuned comparison, *where HAMSTA is still tuned only cross-domain*, also shows that HAMSTA has the best cross-domain performance with a μ_{norm} score of 236.98, and with the 2nd best per-domain tuned move acceptance method being SA with a μ_{norm} score of 253.76. The results of performing a Friedman's test on the per-domain tuned move acceptance methods including HAMSTA shows that there is a significant performance difference between the tested cross-domain tuned move acceptance methods $\big(p = 0.00, \chi^2(8) = 2660.64\big)$. Post-hoc analysis shows that while HAMSTA has the best cross-domain performance, it does not statistically significantly differ from SA, SARH, AILLA, or IE with respective μ_{norm} scores of 253.76, 323.70, 382.70, and 405.74. Furthermore, performing a Friedman's test and post-hoc analysis for each problem domain, showing the significance in performance difference between move acceptance methods for solving each domain independently, shows that HAMSTA does not perform significantly worse than any other move acceptance method for any of the tested problem domains - both for cross-domain and per-domain tuned comparisons.

Table 5.2: ν_{norm} and μ_{norm} scores and standard deviations of $f_{norm}(s)$ values for each move acceptance method compared for each problem domain and finally cross-domain for both per-domain tuned and cross-domain tuned move acceptance methods where HAMSTA is tuned cross-domain in both comparisons. ν_{norm} and μ_{norm} scores are stylised **bold** to indicate that a move acceptance method does not perform statistically significantly different (worse) than the best performing move acceptance method for each problem domain (and finally cross-domain) based on a Friedman test (CI = 95%) with post-hoc analysis based on the Bonferroni correction procedure. The χ^2 and p values are the results of performing such Friedman test. The cross-domain μ_{norm} is the sum of ν_{norm} scores over all problem domains, and the cross-domain standard deviation is the standard deviation of the standard deviations over all domains.

Cross-domain Tuned		BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP	Cross-domain
IE ($\pi_{\mathtt{xd}}$)	ν_{norm}	**2.66**	**16.12**	112.06	**16.74**	**61.53**	**15.49**	155.00	**20.68**	**16.33**	**416.61**
	Std.	0.0162	0.0536	0.2378	0.0415	0.1983	0.0685	0.0000	0.0590	0.0446	0.342
AILLA ($\pi_{\mathtt{xd}}$)	ν_{norm}	**26.39**	**17.32**	**78.02**	**2.51**	**66.73**	106.54	**62.10**	133.79	**7.24**	**500.64**
	Std.	0.0613	0.0613	0.2960	0.0140	0.2403	0.2518	0.3881	0.0893	0.0338	0.346
TA ($\pi_{\mathtt{xd}}$)	ν_{norm}	84.51	**91.64**	112.06	**16.75**	**82.64**	**38.41**	155.00	110.89	64.66	756.58
	Std.	0.1552	0.2407	0.2378	0.0414	0.2098	0.1619	0.0000	0.3550	0.2144	0.325
GD ($\pi_{\mathtt{xd}}$)	ν_{norm}	104.57	**90.06**	**48.03**	111.60	**87.88**	**103.28**	152.51	99.80	111.05	908.79
	Std.	0.3773	0.3037	0.3278	0.3924	0.2102	0.3754	0.0536	0.3375	0.3926	0.365
AILTA ($\pi_{\mathtt{xd}}$)	ν_{norm}	**23.88**	**78.00**	111.81	**16.74**	**84.61**	**7.49**	155.00	**24.95**	**9.80**	512.29
	Std.	0.0458	0.2235	0.2388	0.0415	0.2327	0.0565	0.0000	0.0783	0.0268	0.350
NA ($\pi_{\mathtt{xd}}$)	ν_{norm}	101.04	120.26	**15.65**	109.52	**96.98**	139.70	**131.02**	134.22	125.78	974.16
	Std.	0.1566	0.1008	0.1207	0.0208	0.1828	0.0780	0.2207	0.0763	0.0659	0.263
SA ($\pi_{\mathtt{xd}}$)	ν_{norm}	87.44	**53.08**	**27.11**	**2.34**	**89.36**	**13.05**	**84.41**	**2.60**	**1.42**	**360.82**
	Std.	0.0987	0.0863	0.1968	0.0096	0.2013	0.0542	0.3532	0.0103	0.0064	0.284
SARH ($\pi_{\mathtt{xd}}$)	ν_{norm}	95.29	120.03	**24.99**	**4.49**	**96.71**	**37.83**	**80.65**	**54.67**	**15.59**	530.25
	Std.	0.0902	0.0982	0.1861	0.0133	0.1811	0.1675	0.3385	0.1361	0.0230	0.299
HAMSTA ($\pi_{\mathtt{xd}}$)	ν_{norm}	**5.98**	**19.52**	**32.00**	**6.41**	**61.57**	**8.03**	**103.63**	**13.62**	**3.43**	**254.19**
	Std.	0.0298	0.0614	0.2609	0.0236	0.1972	0.0575	0.3582	0.0597	0.0094	0.264
$\chi^2(8)$		1064.75	1043.91	966.8	1190.06	231.76	953.16	822.79	984.38	1164.97	3232.93
p-value		1.57×10^{-224}	4.95×10^{-220}	2.18×10^{-203}	1.35×10^{-251}	1.25×10^{-45}	1.92×10^{-200}	2.52×10^{-172}	3.507×10^{-207}	3.55×10^{-246}	0
Per-domain Tuned		BP	FS	PS	SAT	TSP	VRP	KP	MAC	QAP	Cross-domain
IE ($\pi_{\mathtt{d}}$)	ν_{norm}	**2.66**	**15.87**	112.50	**16.88**	**54.54**	**15.37**	**155.00**	**17.17**	**15.76**	**405.74**
	Std.	0.0162	0.0541	0.2364	0.0420	0.2072	0.0679	0.0000	0.0568	0.0463	0.3429
TA ($\pi_{\mathtt{d}}$)	ν_{norm}	**12.09**	**19.61**	**23.68**	**16.88**	**76.82**	**5.27**	**152.20**	**70.93**	**5.22**	**382.70**
	Std.	0.0724	0.0623	0.1546	0.0420	0.2346	0.0329	0.0589	0.3844	0.0215	0.3406
GD ($\pi_{\mathtt{d}}$)	ν_{norm}	103.73	**90.55**	**48.20**	111.92	**82.89**	**103.27**	**152.19**	**97.17**	111.59	901.51
	Std.	0.3869	0.3020	0.3261	0.3924	0.2197	0.3723	0.0663	0.3398	0.3919	0.3675
NA ($\pi_{\mathtt{d}}$)	ν_{norm}	101.09	120.21	**17.86**	109.99	**92.84**	139.55	**129.06**	132.13	125.88	968.61
	Std.	0.1564	0.1009	0.1264	0.0192	0.1881	0.0792	0.2407	0.0687	0.0675	0.2611
SA ($\pi_{\mathtt{d}}$)	ν_{norm}	**11.02**	**20.69**	**20.51**	**2.67**	**72.81**	**28.31**	**69.44**	**17.36**	**10.95**	**253.76**
	Std.	0.0626	0.0648	0.1491	0.0086	0.2217	0.1288	0.3873	0.0562	0.0168	0.2272
SARH ($\pi_{\mathtt{d}}$)	ν_{norm}	93.53	**20.95**	**21.51**	**1.57**	**71.44**	**28.31**	**68.14**	**7.47**	**10.78**	**323.70**
	Std.	0.0868	0.0592	0.1578	0.0071	0.2217	0.1288	0.3853	0.0375	0.0158	0.2614
HAMSTA ($\pi_{\mathtt{xd}}$)	ν_{norm}	**5.98**	**19.28**	**33.12**	**6.52**	**54.49**	**7.93**	**97.00**	**9.89**	**2.77**	**236.98**
	Std.	0.0298	0.0618	0.2671	0.0240	0.2075	0.0560	0.4002	0.0491	0.0102	0.2608
$\chi^2(6)$		828.85	537.98	452.64	944.64	162.03	610.82	574.41	612.57	877.64	2660.64
p-value		8.97×10^{-176}	5.50×10^{-113}	1.33×10^{-94}	8.40×10^{-201}	2.20×10^{-32}	1.08×10^{-128}	7.69×10^{-121}	4.52×10^{-129}	2.57×10^{-186}	0

Looking at the performance of HAMSTA compared to the other move acceptance methods tested for each of the problems, HAMSTA is the only move acceptance method from the cross-domain tuned comparison that does not perform significantly worse than any other method over all 9 domains. HAMSTA is closely followed by SA; however, SA performs significantly worse than the best method for solving instances from the Bin Packing problem. When HAMSTA is compared to the per-domain tuned move acceptance methods, HAMSTA is also able to perform not significantly worse than the best method over all 9 domains. This is also true for SA and TA, however, as discussed above, their relative cross-domain performance, despite being re-tuned for each problem, is not as good as HAMSTA tuned cross-domain.

An empirical cumulative distribution function (ECDF) plot is given in Figure 5.9 and shows the cross-domain performance of the per-domain and cross-domain tuned move acceptance methods, and with HAMSTA being shown by the green line. The ECDF plot shows the cumulative frequency of $f_{norms}(s)$ values obtained by each move acceptance method calculated from the entire set of results including both per-domain and cross-domain results configurations. The area under each line ($\int_0^1 ecdf(m)dm$ where m is the move acceptance method) thus represents the cross-domain performance of each move acceptance method where the ultimate method (u), finding the best solution at each trial, would have an area of $\int_0^1 ecdf(u)du = 1.0000$. The area under each line is inversely proportional to the μ_{norm} score of each respective move acceptance method such that $\int_0^1 ecdf(m)dm = 1 - \dfrac{\mu_{norm}(m)}{n}$ where n is the total number of trials. Note that the values given for the area under the lines for the ECDF plot in this study is calculated as the lower Darboux integral with a sub-interval width of 1.00×10^{-6}.

While HAMSTA has the best cross-domain performance, $\int_0^1 ecdf(\texttt{HAMSTA})d\texttt{HAMSTA} = 0.8170$, the ECDF plot also shows that SA (π_{xd}) can obtain a higher frequency of the highest quality solutions, although across the full spectrum of results HAMSTA has the clear best cross-domain performance. The ecdf areas are given in Table 5.3 which shows the cross-domain performance of each move acceptance method when compared to each of the other methods. HAMSTA is the best performing move acceptance method, $\int_0^1 ecdf(\texttt{HAMSTA}(\pi_{\texttt{xd}}))d\texttt{HAMSTA}(\pi_{\texttt{xd}}) = 0.8170$, despite being tuned only cross-domain, with per-domain tuned variants of SA, $\int_0^1 ecdf(\texttt{SA}(\pi_{\texttt{d}}))d\texttt{SA}(\pi_{\texttt{d}}) = 0.8047$, and SARH $\int_0^1 ecdf(\texttt{SARH}(\pi_{\texttt{d}}))d\texttt{SARH}(\pi_{\texttt{d}}) = 0.7542$ coming 2nd and 3rd respectively. HAMSTA outperforms the first existing cross-domain tuned move acceptance method, SA, which is 4th with $\int_0^1 ecdf(\texttt{SA}(\pi_{\texttt{xd}}))d\texttt{SA}(\pi_{\texttt{xd}}) = 0.7400$.

Figure 5.10 also show the $f_{norms}(s)$ values obtained by each move acceptance method calculated from the entire set of results including both per-domain and cross-domain results configurations in

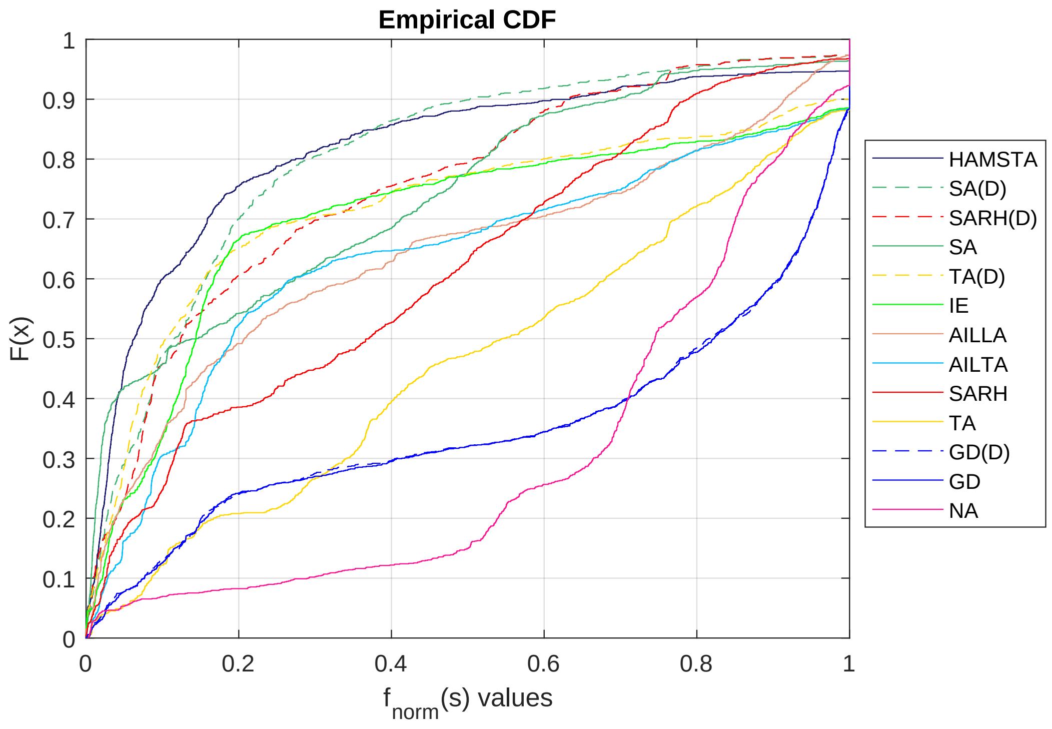


Figure 5.9: An empirical cumulative distribution function using the $f_{norm}(s)$ values from all 1395 trials spanning 45 instances from 9 problem domains, illustrating the cross-domain performance of each cross-domain tuned move acceptance method, and where possible re-tuned for each problem domain. An ideal cross-domain search algorithm will obtain the best solution for each problem being solved; the objective in this case is therefore to maximise the cumulative distribution while minimising the μ_{norm} values. Solid lines represent cross-domain tuned move acceptance methods, whereas dashed lines represent those tuned per-domain. Each move acceptance method is ordered in the legend based on their cross-domain performance as the area under the respective frequency curve.

boxplot form. The black boxplots indicate that the respective move acceptance method uses its cross-domain tuned parameter configuration whereas the blue boxplots indicate that the respective move acceptance method uses its per-domain tuned parameter configurations. Each move acceptance method and configuration is ordered from worst (top) to best (bottom) by their mean average $f_{norm}(s)$ values and shows that HAMSTA is the only cross-domain tuned move acceptance method to outperform all other per-domain tuned move acceptance methods.

5.5 Trace Analysis

The plots shown in Figure 5.11 show the progress traces for HAMSTA when using the cross-domain parameter configuration (top) and domain-specific parameter configurations (bottom) on

Table 5.3: Areas under each move acceptance method's ECDF plot representing their cross-domain performance (higher is better).

Move Acceptance Method (m)	Tuning Approach	$\int_0^1 ecdf(m)dm$
HAMSTA	$\pi_{\mathtt{xd}}$	0.8170
SA	$\pi_{\mathtt{d}}$	0.8047
SARH	$\pi_{\mathtt{d}}$	0.7542
SA	$\pi_{\mathtt{xd}}$	0.7400
TA	$\pi_{\mathtt{d}}$	0.7192
IE	$\pi_{\mathtt{xd}}$	0.7012
AILLA	$\pi_{\mathtt{xd}}$	0.6419
AILTA	$\pi_{\mathtt{xd}}$	0.6327
SARH	$\pi_{\mathtt{xd}}$	0.6191
TA	$\pi_{\mathtt{xd}}$	0.4590
GD	$\pi_{\mathtt{d}}$	0.3498
GD	$\pi_{\mathtt{xd}}$	0.3487
NA	$\pi_{\mathtt{xd}}$	0.3015

an instance chosen from each of the benchmark problem domains. With the exception of the Quadratic Assignment problem, the behaviour of HAMSTA guiding the search of the search space of each problem does not differ significantly between the cross-domain and per-domain configurations. Comparing the behaviours of HAMSTA using its cross-domain parameter configuration to show its versatility for solving characteristically different COPs shows how well it can adapt to the problem being solved.

Bin Packing, Flow Shop and Travelling Salesman problems were shown to be solved the best using IE move acceptance in the work from Section 4.2, and the plots for HAMSTA to solve these instances, Figures 5.11a, 5.11b, and 5.11e respectively, illustrate this behaviour in general. Note that while the search *appears* to be stuck in plots 5.11a and 5.11e, this is not the case as the relative differences between each solution are very small compared to the initial improvements made towards the start of the search. For solving flow shop on the other hand, Figure 5.11b illustrates the scenario where a local optima is reached (as indicated by the best solution found marker) and after a period of non-improvement, HAMSTA enters the AM stage to force exploration of the search space. The plots however show that in this case, the search stays in the same optima and subsequent HTA stages converge on the same solution.

Personnel Scheduling problems on the other hand were shown to be solved best by a Naïve Acceptance strategy, and the relevant plot for HAMSTA, shown in Figure 5.11c, shows this expected search trajectory. In this plot, it can be seen that the initial number of iterations to find the best solution are very small (7), and this leads to HAMSTA in the subsequent HTA stages to frequently switch to AM stages throughout the search after a small number of non-improving iterations. The

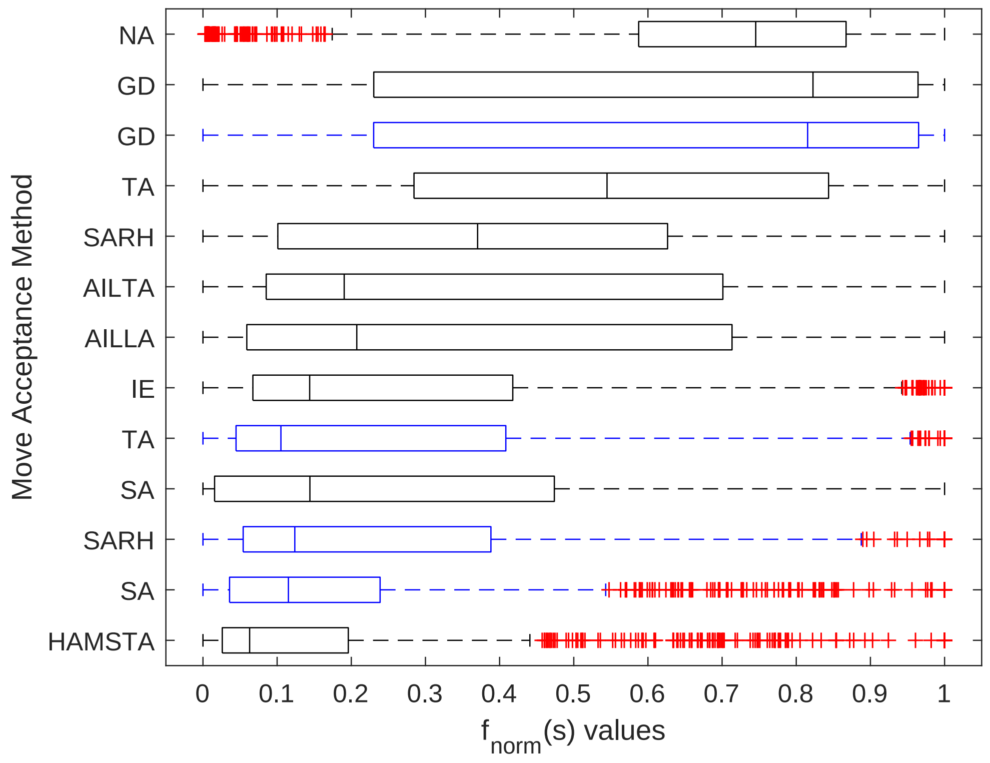


Figure 5.10: Boxplots showing the distribution of $f_{norm}(s)$ scores (lower is better) for all move acceptance methods, including those tuned per-domain (blue) and cross-domain (black) across all 1395 trials from the 45 instances spanning the 9 benchmark problem domains. Move acceptance methods are ordered by their mean $f_{norm}(s)$ scores where smaller is better. '+' marks symbolise statistical outliers according to either $f_{norm}(s) > q_3 + 1.5 \times (q_3 - q_1)$ or $f_{norm}(s) < q_1 - 1.5 \times (q_3 - q_1)$ where q_1 and q_3 are the 25th and 75th sample data percentiles.

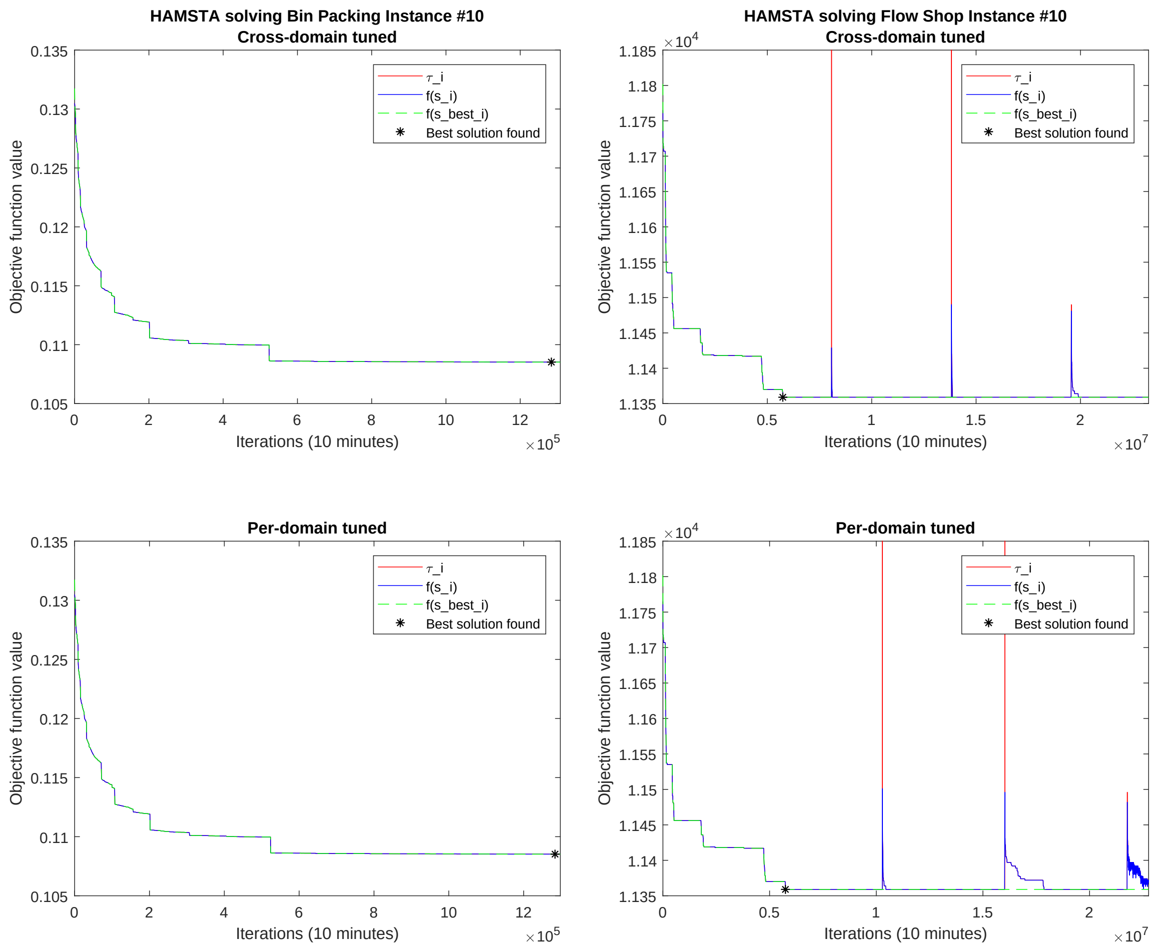
mechanism to determine threshold values as the difference between the accepted solution and the best solution found in the current HTA stage also helps to promote exploration of the search space since the difference in objective values between the best solution found in previous stages, and that in the AM stage is very large. Together, both mechanisms promote a random walk behaviour of HAMSTA when solving the Personnel Scheduling problem.

The progress trace for HAMSTA solving Max Cut and SAT problems, shown in Figures 5.11g and 5.11d, shows an interesting search trajectory and demonstrates how the HTA stages allow the search to maintain an adequate level of exploration while promoting exploitation while the best solution found in the given stage is being improved. Furthermore, between different HTA stages, the adaptation of η can be observed by variable lengths of individual HTA stages.

The progress plot for HAMSTA solving the QAP problem highlights the importance of having an increased setting for η_0 with the cross-domain plot ($\eta_0 = 0.01$) not allowing enough time to converge on a good solution and frequently revisiting AM stages, and with the per-domain plot ($\eta_0 = 0.10$) taking $\approx 60\%$ of the total search time before revisiting the AM stage for the second time.

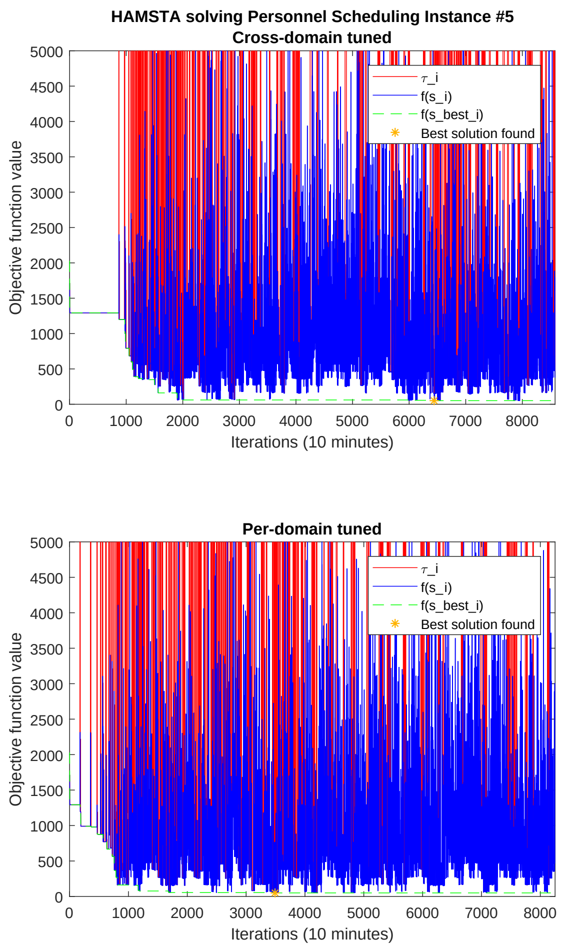
Previous chapters in this work showed that KP instances benefit from frequently accepting a single indiscriminate worse move and improving from that point in the search space. HAMSTA replicates this behaviour as highlighted by the abundance of "infinite" threshold values (τ_i) relating to the high number of AM stages (1976 and 3611 for cross-domain and per-domain configurations), though these values are clipped in the plots.

Over all the domains, HAMSTA is able to demonstrate its ability to adapt to solving different problems. When beneficial, HAMSTA is able to sustain the acceptance of improving or equal moves. Even when tuned cross-domain, it is apparent that the majority of moves are equal or improving only and this allows HAMSTA to perform well for solving BP, FS, TSP and VRPTW problems. For other domains where acceptance of indiscriminate worse moves are beneficial such as for PS and KP, HAMSTA can adapt to solving these problems by having a large setting for *cnilr* and high threshold values due to the values learnt between the AM and HTA stages allowing for HAMSTA to behave as a random walk. For other problems where a balance is needed between exploitation and exploration, HAMSTA can be seen to adapt its internal parameters very differently and use meaningful threshold values throughout the search to improve the best solution over time by increasing exploitation of the search when the best solution can be improved, and facilitation exploration when the best solution cannot be improved. Furthermore, the threshold values learned online from the instances being solved means that they are never too high that the search gets lost exploring the search space in a random walk, and is never too low that subsequent HTA stages cannot improve the best solution.

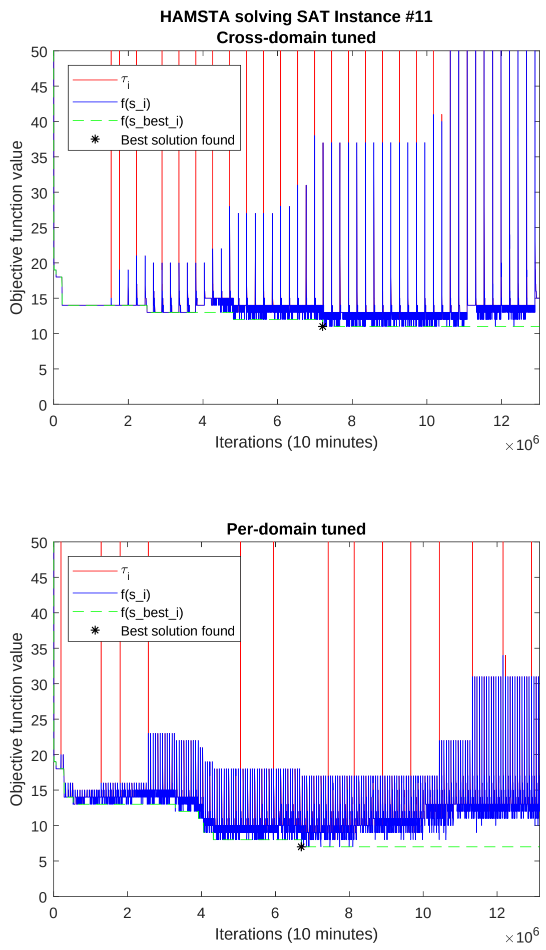


(a) (b)

Figure 5.11: Progress traces of HAMSTA tuned cross-domain (top), $\left(\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 1\}\right)$, and per-domain (bottom), solving an instance from each of the problem domains from the benchmark suite where; (a) is HAMSTA solving instance #10 from the BP domain $\left(\pi_d^{\text{HAMSTA}}(\text{BP}) = \{\epsilon = 0.20, \eta 0 = 0.01, \omega_0 = 1, \omega^+ = 1\}\right)$, and (b) is HAMSTA solving instance #10 of the FS domain $\left(\pi_d^{\text{HAMSTA}}(\text{FS}) = \{\epsilon = 0.20, \eta 0 = 0.20, \omega_0 = 1, \omega^+ = 0\}\right)$.

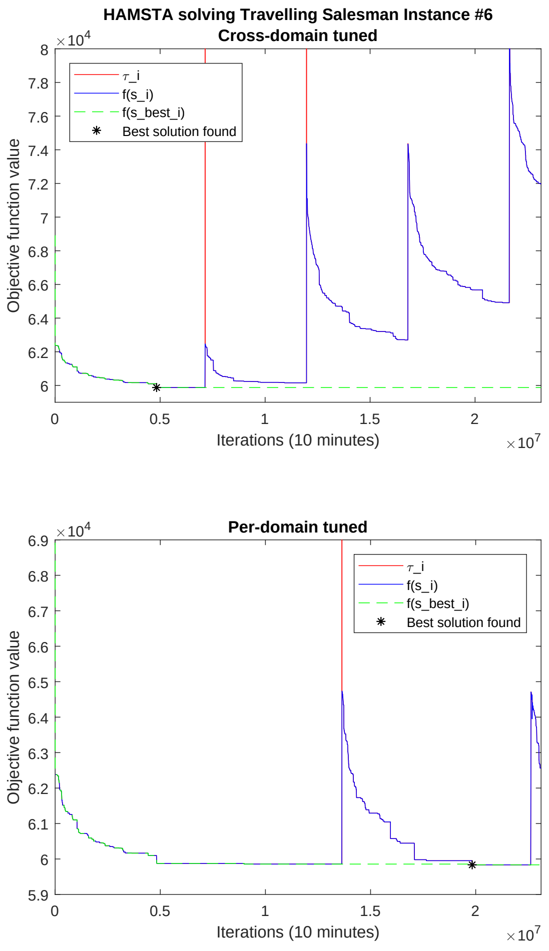


(c)

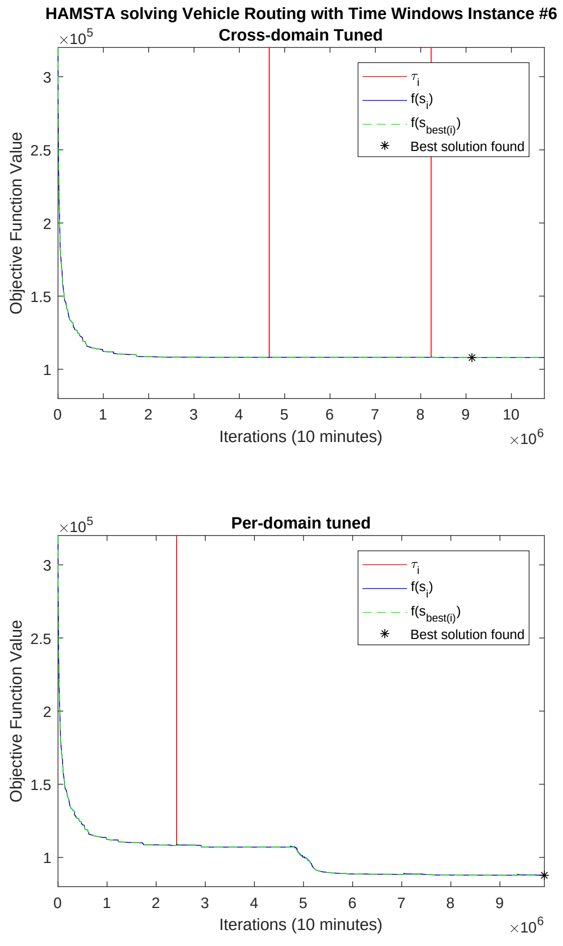


(d)

Figure 5.11: Progress traces of HAMSTA tuned cross-domain (top), $\left(\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 1\}\right)$, and per-domain (bottom), solving an instance from each of the problem domains from the benchmark suite where; (c) is HAMSTA solving instance #5 from the PS domain $\left(\pi_d^{\text{HAMSTA}}(\text{PS}) = \{\epsilon = 0.02, \eta_0 = 0.02, \omega_0 = 2, \omega^+ = 2\}\right)$, and (d) is HAMSTA solving instance #11 of the SAT domain $\left(\pi_d^{\text{HAMSTA}}(\text{SAT}) = \{\epsilon = 0.01, \eta_0 = 0.05, \omega_0 = 1, \omega^+ = 1\}\right)$.



(e)



(f)

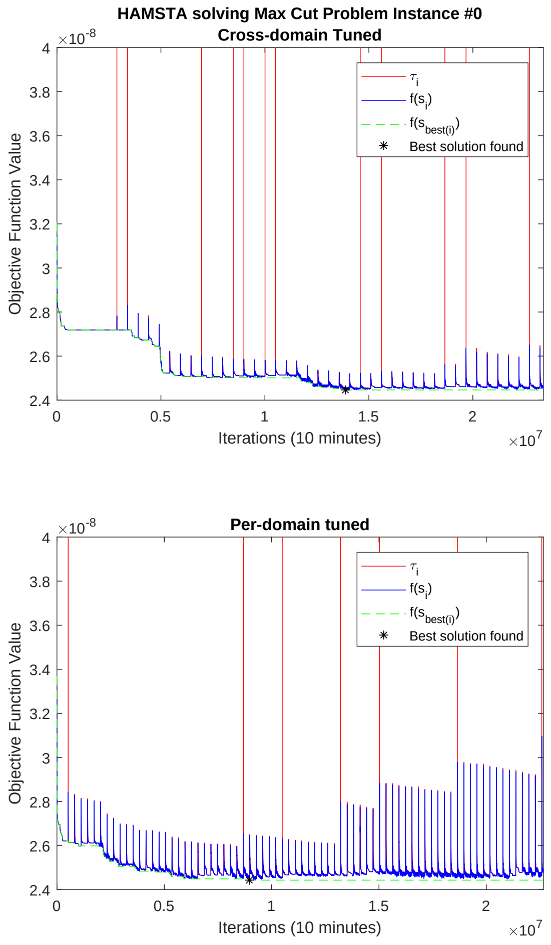
Figure 5.11: Progress traces of HAMSTA tuned cross-domain (top), $\left(\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 1\}\right)$, and per-domain (bottom), solving an instance from each of the problem domains from the benchmark suite where; (e) is HAMSTA solving instance #6 from the TSP domain $\left(\pi_d^{\text{HAMSTA}}(\text{TSP}) = \{\epsilon = 0.20, \eta_0 = 0.20, \omega_0 = 1, \omega^+ = 1\}\right)$, and (f) is HAMSTA solving instance #6 of the VRPTW domain $\left(\pi_d^{\text{HAMSTA}}(\text{VRPTW}) = \{\epsilon = 0.01, \eta_0 = 0.10, \omega_0 = 4, \omega^+ = 3\}\right)$.

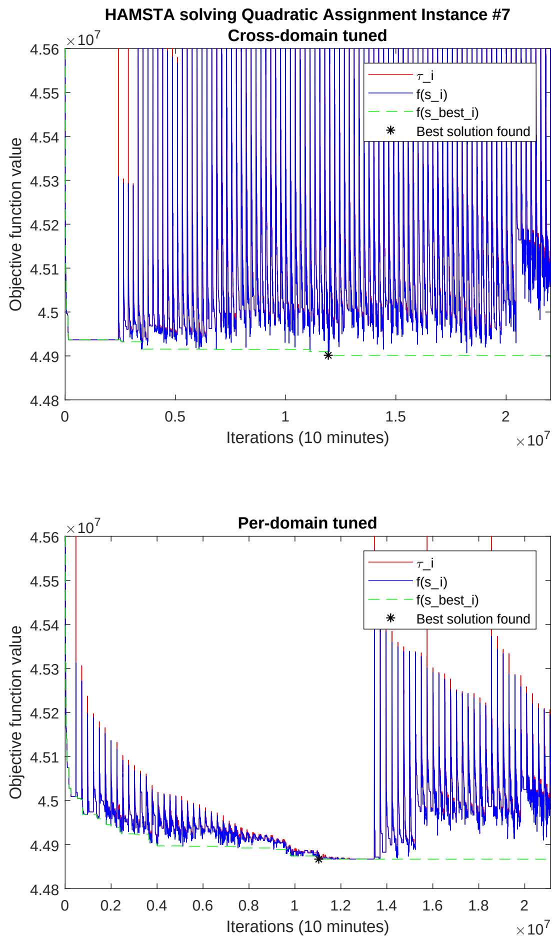
Table 5.4: Comparison of the number of IE, AM, and HTA stages, the list lengths (δ) and initial settings for the maximum number of consecutive non-improving list repetitions (η) used during an execution of HAMSTA given a 10 nominal minute computational budget using its cross-domain and per-domain parameter settings across a randomly selected instance from each problem domain.

Domain	Instance ID	Tuning	IE Stages	AM Stages	HTA Stages	δ	η_{init}
BP	10	$\pi_{\mathtt{xd}}$	1	0	0	-	-
BP	10	$\pi_{\mathtt{d}}$	1	0	0	-	-
FS	10	$\pi_{\mathtt{xd}}$	1	3	3	5742245	1
FS	10	$\pi_{\mathtt{d}}$	1	3	3	5742245	1
PS	5	$\pi_{\mathtt{xd}}$	1	179	179	7	12
PS	5	$\pi_{\mathtt{d}}$	1	140	140	7	22
SAT	11	$\pi_{\mathtt{xd}}$	1	25	25	227164	1
SAT	11	$\pi_{\mathtt{d}}$	1	14	14	64001	11
TSP	6	$\pi_{\mathtt{xd}}$	1	4	4	4831558	1
TSP	6	$\pi_{\mathtt{d}}$	1	2	2	8989696	1
VRPTW	6	$\pi_{\mathtt{xd}}$	1	2	2	3571521	1
VRPTW	6	$\pi_{\mathtt{d}}$	1	1	1	2314027	1
KP	8	$\pi_{\mathtt{xd}}$	2	1976	1974	61	70
KP	8	$\pi_{\mathtt{d}}$	2	3611	3610	63	69
MAC	0	$\pi_{\mathtt{xd}}$	1	4	4	522480	5
MAC	0	$\pi_{\mathtt{d}}$	1	10	10	984355	1
QAP	7	$\pi_{\mathtt{xd}}$	1	28	28	134852	2
QAP	7	$\pi_{\mathtt{d}}$	1	6	6	123681	18

Run-time data for HAMSTA was recorded alongside the above progress traces tracking the number of IE, AM, and HTA stages. These are summarised in Table 5.4. Between each problem domain, the adaptability of HAMSTA can be seen by the variation in the number of stages, δ (the estimated number of iterations to encounter a local optima), and η_{init} (the initial target for the consecutive number of list repetitions where the best solution found during a given HTA stage has not improved before the stage terminates and transitions to an AM stage) given an identical parameter configuration (π_{xd}). The setting of ϵ influences when HAMSTA believes the search is stuck in a local optimum during the IE stage and this can influence the value that is found for δ. HAMSTA encounters settings of δ which are orders of magnitudes different between different domains and instances, and depending on the time taken to find the local optimum, this can also effect the initial setting of η which can also vary by orders of magnitude between domains and their instances.

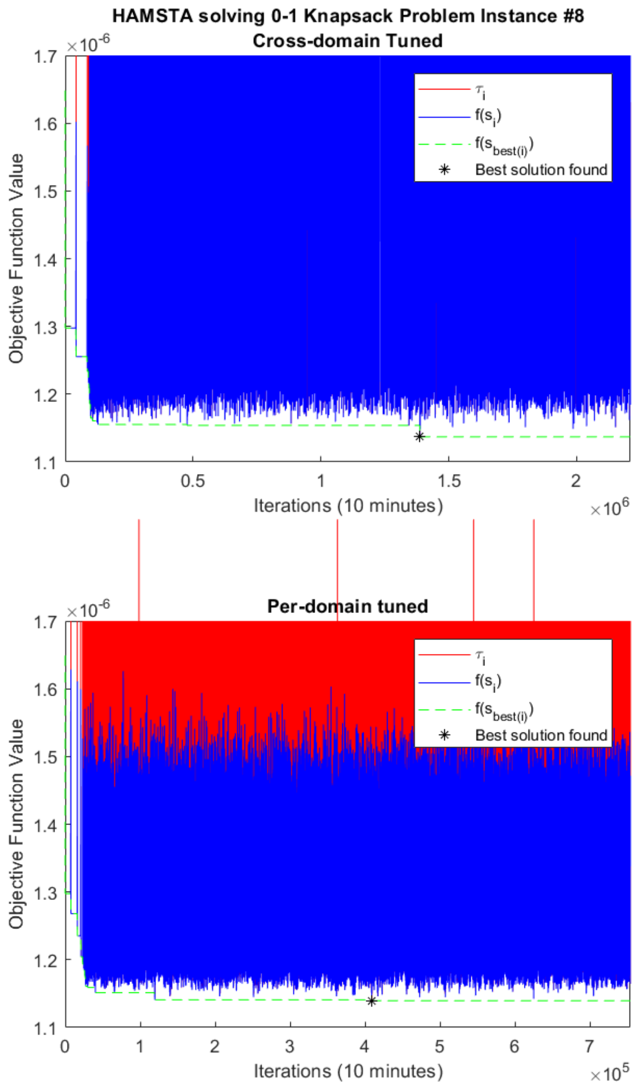
When comparing HAMSTA between each parameter tuning approach for the same domains, the different parameter configurations between the cross-domain configuration ($\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \omega_0 = 0.01, \omega^+ = 1, \eta_0 = 1\}$) and per-domain configurations, as shown in Table 5.1, HAMSTA can be seen to have varied outcomes on the values found for δ. Increasing the setting for ϵ (for these instances) sometimes did not affect the value of δ, such as in FS, and this indicates that





(g) (h)

Figure 5.11: Progress traces of HAMSTA tuned cross-domain (top), $\left(\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 1\}\right)$, and per-domain (bottom), solving an instance from each of the problem domains from the benchmark suite where; (g) is HAMSTA solving instance #0 from the MAC domain $\left(\pi_d^{\text{HAMSTA}}(\text{MAC}) = \{\epsilon = 0.01, \eta_0 = 0.10, \omega_0 = 5, \omega^+ = 2\}\right)$, and (h) is HAMSTA solving instance #7 of the QAP domain $\left(\pi_d^{\text{HAMSTA}}(\text{QAP}) = \{\epsilon = 0.01, \eta_0 = 0.10, \omega_0 = 1, \omega^+ = 0\}\right)$.



(i)

Figure 5.11: Progress traces of HAMSTA tuned cross-domain (top), $\left(\pi_{xd}^{\text{HAMSTA}} = \{\epsilon = 0.10, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 1\}\right)$, and per-domain (bottom), solving an instance from each of the problem domains from the benchmark suite where; (i) is HAMSTA solving instance #8 from the KP domain $\left(\pi_d^{\text{HAMSTA}}(\text{KP}) = \{\epsilon = 0.02, \eta_0 = 0.01, \omega_0 = 1, \omega^+ = 0\}\right)$.

even different instances from the same domain can behave differently. That is, at least one other instance must have benefited from an increased ϵ setting resulting in a higher value of δ. Increasing the setting for ϵ also resulted in much higher values, such as in TSP and illustrates the expected behaviour. Decreasing ϵ on the other hand has the advantage of reducing the time waiting in a local optimum, but with the trade off of potentially ending prematurely resulting in decreased values for δ. The values for δ for PS and VRP instances remained the same whereas for KP, MAC and QAP, the values for δ were slightly decreased. For SAT on the other hand, the value for δ was significantly less, and this is despite the setting of ϵ being the same for SAT, VRP, KP, MAC and QAP. For all domains but VRP, an equal or increased initial value for η, either facilitated directly by an increased η_0 setting or decreased value for δ, was used when HAMSTA was tuned per-domain when compared to its cross-domain configuration. In the case of the VRP instance, an equal setting of η_0 and equal value of δ for both cross-domain and per-domain configurations indicates that the variation of η_{init} is due to the variation in the time taken to reach the best solution found during the IE stage. Referring back to the trace for the QAP problem instance in Figure 5.11h, the increase of η_{init} can be seen to have a significant effect on the search trajectory, with almost 2/3rds of the computational budget elapsing before the second AM stage. A similar effect is seen for solving the MAC problem instance in Figure 5.11g where the search goes from being a random walk to an effective search trajectory where exploitation can take over to reach regions of local optima.

HAMSTA adapts to the search landscapes by adapting η throughout the search process, and this effects the total number of AM and HTA stages given the pre-defined computational budget. At one end of the spectrum, BP constantly improves the solution-in-hand over time since the local search metaheuristic framework requires no exploration to find a high quality solution and thus no AM or HTA stages are ever executed. On the other hand, HAMSTA when tuned cross-domain for solving the KP problem executes a total of 11657 AM stages, and 11655 stages. Given a per-domain tuning approach, these values are one order of magnitude higher. With the exception of the KP domain, all other domains were solved more effectively when using a lower number of AM and HTA stages when compared to HAMSTA's cross-domain configuration. An example of where HAMSTA begins and ends with a high η value is with the per-domain tuned MAC problem instance. Initially, η is calculated to be 196. Within the first HTA stage, the best solution is improved and requires 82 list repetitions to reach it. Thus, in the second HTA stage, η is set to 114. In this stage, the best solution is improved again using 42 list repetitions and set to 72 in the subsequent stage. The next stage is unable to improve the best solution found so far, and so η takes its previous value of 72. This process continues and the adaptation of η throughout the search process can be observed

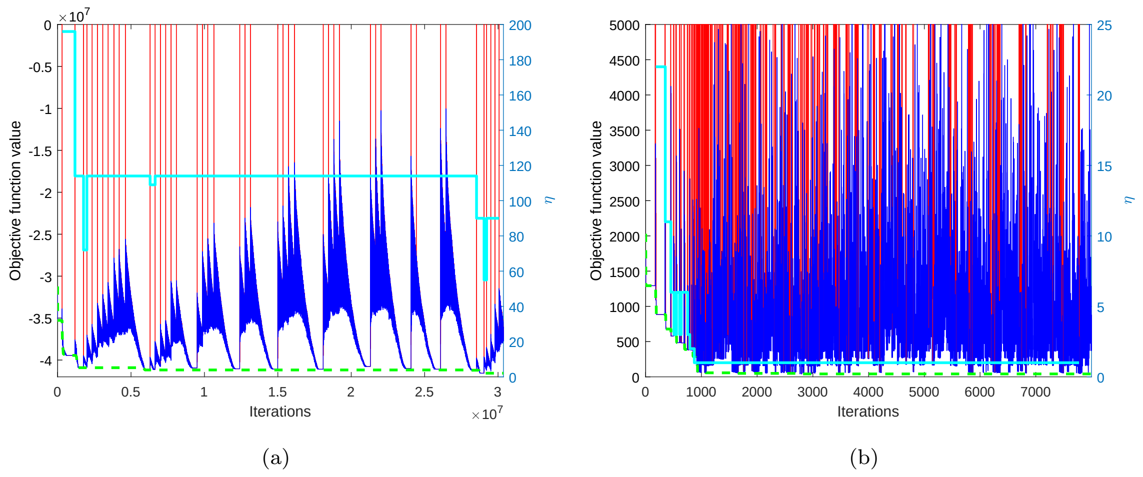


Figure 5.12: Progress trace of HAMSTA tuned per-domain solving; (a) instance #0 of the MAC domain, and (b) instance #5 of the PS domain, highlighting the adaptation of η (cyan) throughout the search process.

in Figure 5.12a. A further example, as shown in Figure 5.12b, shows the values of η converge on 1 over time as HAMSTA solves the PS problem instance which appears to favour a random walk search trajectory.

Examples of the adaptation of omega (ω) are shown in Figure 5.13a and Figure 5.13b for HAMSTA solving MAC instance #0 and PS instance # 5. For solving the MAC instance, ω can be seen to increase at each HTA stage to higher values before the search is able to find a good quality region in the search landscape. In some cases, this is not as good as the best solution found so far, and ω continues to increase before finding other feasibly good regions. Eventually, this region is good enough such that the best solution found so far is improved, and ω is reset to its initial value. For solving the PS instance, ω initially increases to small values (up to 10) for the first 1000 iterations. From then on, higher ω values are required to escape the current search landscape region to find better quality solutions. ω can then be seen to increase to 36 and 48 to find better quality solutions, before ω seemingly increases for the foreseeable future while a random walk of the search space is performed in an attempt to find better quality solutions in other regions of the search space.

5.6 A Comparison to the State-of-the-Art

In this work, the cross-domain performance of move acceptance methods have been evaluated under a single point-based stochastic local search metaheuristic framework. Over the years, hyper-

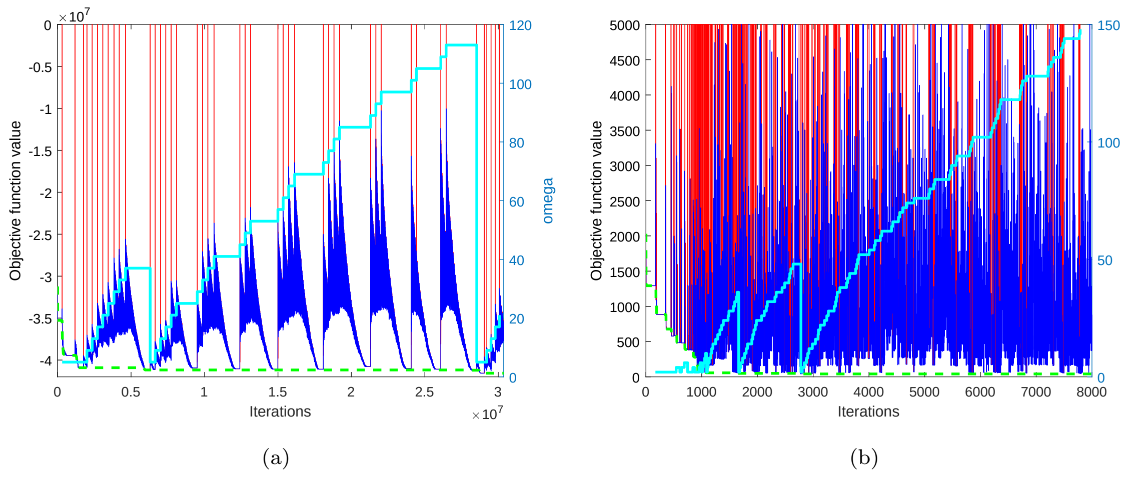


Figure 5.13: Progress trace of HAMSTA tuned per-domain solving; (a) instance #0 of the MAC domain, and (b) instance #5 of the PS domain, highlighting the adaptation of ω (cyan) throughout the search process.

heuristics have emerged as the go to state-of-the-art search method for solving the cross-domain search problem. In this section, the performance of HAMSTA under the local search metaheuristic framework as defined in Chapter 3 (denoted LSM-R for the restricted set of heuristics) is compared to the current state-of-the-art hyper-heuristic search methods (denoted HH) for cross-domain search. Two other local search metaheuristics are included for completeness operating on the same local search metaheuristic framework, but with access to the full set of low-level operators (LSM-F) including mutation, local search, and ruin-recreate.

It should be noted here that the move acceptance method from the best hyper-heuristic method from the CHeSC 2011 competition (AdapHH) was used in the above studies under the local search metaheuristic framework as AILLA. The results below are obtained from comparing HAMSTA-LSM-R to the top-three hyper-heuristics that utilise elaborate heuristic selection methods for cross-domain search from different search frameworks, and two other local search metaheuristic methods which make use of all non-crossover heuristics. These include AdapHH [120], a single point-based selection hyper-heuristic that applies either one or two low-level heuristics in succession to generate a candidate solution, EPH [178], a population-based search method based on evolutionary programming embedding co-evolution of the solutions and settings for low-level heuristics, and SSHH [60], a single point-based hyper-heuristic that uses a hidden Markov model to construct and apply sequences of low-level heuristics. The two local search metaheuristic methods that have access to the full set of heuristics include AM-LMS-F and IE-LSM-F, and both select a low-level heuristic at random using the AM and IE move acceptance methods, respectively. AM-LSM-F accepts all candidate moves whereas IE-LSM-F accepts all non-worsening moves. In the performance comparisons, the six CHeSC 2011 benchmark problem domains are used [20].

The results from performing a Friedman test, as shown in Table 5.5, on HAMSTA-LSM-R and the other methods shows that the best cross-domain search methods are all three hyper-heuristics embedding heuristic selection mechanisms ($\chi^2(7) = 3038.58$; $p = 0$). Observation of the mean ranks would suggest that having a set of specialised low-level heuristics allows such search methods to perform better than a local search metaheuristic framework containing a minimal set of perturbative operators with IE-LSM-F and IE-LSM-R having ranks of 134.1 and 140.2 respectively. A Wilcoxon Signed Rank test comparing these two methods only shows that IE-LSM-F significantly outperforms IE-LSM-R with a one-tailed test showing a significant result ($p = 2.722 \times 10^{-34}$). The mean ranks also suggest that while HAMSTA-LSM-R under the local search metaheuristic framework is not able to perform as well as the hyper-heuristic methods employing heuristic selection strategies, it does perform better than both local search metaheuristics that contain the full set of heuristics. Subsequent Wilcoxon Signed Rank test shows that actually, HAMSTA-LSM-R is able to outperform AM-LSM-F ($p = 7.335 \times 10^{-11}$) but IE-LSM-F actually outperforms HAMSTA-LSM-R ($p = 1.719 \times 10^{-4}$).

Table 5.5: Friedman test comparing the cross-domain performance of HAMSTA to state-of-the-art methods with n_0 that all results are from the same distribution at CI = 95%. The values are the mean ranks (lower is better) of the aforementioned test. The best search method (as chosen as that with the lowest mean rank), and those which do not statistically significantly differ from the best, for each domain being stylised bold.

Search Method	AdapHH	SSHH	EPH	AM-LSM-F	IE-LSM-F	HAMSTA-LSM-R	IE-LSM-R	$\chi^2(7)$	p
Framework	HH	HH	HH	LSM-F	LSM-F	LSM-R	LSM-R		
Cross-domain ranks	**59.5**	**50.4**	**75.5**	173.9	136.3	126.3	141.1	3038.58	0

The cross-domain μ_{norm} scores are shown in Table 5.6 and are indicative of the test statistics discussed above. What these results show is that while increasing the number of low-level heuristics does improve the cross-domain performance of the search method using them, there must be some form of move acceptance for them to perform well. That is, the local search metaheuristics with a full heuristic set accepting all moves (AM-LSM-F) performs worse than both HAMSTA-LSM-R and IE-LSM-R under the local search metaheuristic framework with the restricted set of operators. Furthermore, the results show that all hyper-heuristic methods with heuristic selection mechanisms (SSHH, AdapHH, and EPH) significantly outperform all random choice search methods (IE-LSM-F, HAMSTA-LSM-R, IE-LSM-R, and AM-LSM-F), irrespective of their move acceptance method.

HAMSTA, as the move acceptance method of a local search metaheuristic, is shown here that it

Table 5.6: Cross-domain μ_{norm} results comparing the cross-domain performance of HAMSTA-LSM-R to state-of-the-art methods.

Search Method	Framework	μ_{norm}
SSHH	HH	76.22
AdapHH	HH	82.26
EPH	HH	115.89
IE-LSM-F	LSM-F	290.52
HAMSTA-LSM-R	LSM-R	310.20
IE-LSM-R	LSM-R	421.12
AM-LSM-F	LSM-F	591.33

cannot perform as good as the state-of-the-art hyper-heuristics. This is despite the reference to [26] claiming that the choice of “move acceptance method significantly affects the performance compared to heuristic selection”. On reflection of the study in [26], their work makes this observation using a small number of simple low-level heuristics and the various hyper-heuristic methods were only applied to solve benchmark function optimisation problems using a binary representation. Later studies, such as [53], made use of a larger set of low-level heuristics, some of which are specialised operators for solving a particular problem. Concerning hyper-heuristics used for solving the cross-domain search problem, real world optimisation problems are tackled, each of which makes use of various representations and specialised low-level heuristics. In those cases, heuristic selection in combination with move acceptance becomes influential on the performance of the overall algorithm.

In conclusion, the choice of move acceptance method *does* significantly affect the cross-domain performance of a heuristic search method. More importantly, the cross-domain performance improvement of a well designed heuristic selection strategy in combination with a move acceptance method significantly outweighs that of the move acceptance method alone when used under a hyper-heuristic framework with a rich set of low-level operators.

5.7 Summary

In this chapter, a novel move acceptance method was sought which when used as a component of a local search metaheuristic, could perform as-good-as the existing move acceptance methods for solving multiple characteristically different COPs, and perform well without requiring expert intervention, such as re-tuning of its parameter settings when solving different problems and their instances. The proposed move acceptance method, called as “HAMSTA” was explained and its cross-domain performance was evaluated and compared to the existing move acceptance methods under the single point-based local search metaheuristic framework. The experimental results show that the cross-domain performance of HAMSTA is not only as-good-as the existing move acceptance methods but is able to outperform them. This is despite HAMSTA utilising a single "cross-domain" parameter configuration whereas those move acceptance methods that were compared to were tuned per-domain and cross-domain, giving them a theoretical advantage over HAMSTA. The outcome and hence the contribution of the work in this chapter is the introduction of a new move acceptance method which is called as "HAMSTA" which is able to outperform the existing move acceptance methods across a set of characteristically different COPs, and without the requirement for expert intervention in the form of parameter tuning after the design stage of the move acceptance method. Revisiting the decision-making dilemma faced by researchers and practitioners when tasked with solving new or unknown COPs, as discussed in Section 1, *"Is there a single algorithm that I can use to solve any given problem, but that does not need to be re-tuned to perform at least as-good-as the existing algorithms?"*, the performance of HAMSTA across a wide range of COPs using a single parameter configuration would suggest that HAMSTA can eliminate this dilemma from the decision-making process as HAMSTA is shown to perform better than existing move acceptance methods, and can do so without the need for re-tuning of its parameters. Hence, HAMSTA should be the move acceptance method of choice for practitioners when solving new and unknown problems as it is empirically shown to produce higher quality solutions on average compared to the existing methods, and can be used by researchers to form benchmarks for their target problems. Moreover, HAMSTA is the only move acceptance method which can outperform the move acceptance methods that are re-tuned for each problem being solved while itself requiring no such efforts. In the following chapter, the work that was conducted in this thesis is concluded. The future research directions based on the contributions of this work are then discussed as well as how these can be further extended, as well as future research focuses that are needed to resolve some criticisms of the broader research area.

Chapter 6

Conclusions and Future Work

6.1 Context

The search methodologies used for tackling real-world combinatorial optimisation problems (COPs) have always been of interest to researchers and practitioners. *Metaheuristics* imposing 'a set of guidelines or strategies' based on a heuristic search framework can be preferred over exact methods due to the fact that many real-world problems are computationally hard to solve and exact methods can fail to produce acceptable solutions in a reasonable time frame. The choice of a suitable move acceptance method to use within a given search framework is important and can significantly affect the performance of the search method being used. When a problem to be solved is *new* or *unseen*, this task is more difficult since a researcher or practitioner will initially not have previous experience, knowledge, or guidance for solving the given problem. Three questions that are frequently asked by researchers and practitioners when given the task of solving a *new* or *unseen* problem include "Which algorithm should I use to solve my problem?", "What parameter settings should I use for my chosen algorithm?", and "Is there another algorithm which will produce better results than the one I have chosen?". An area of research concerns the development of high-level, general-purpose search methodologies known as cross-domain search methods. Cross-domain search is the term used to describe the high-level issue of devising a single search method which is able to solve multiple characteristically different COPs to a high quality given a pre-defined computational budget with the least, but preferably without, expert intervention or modification. State-of-the-art cross-domain search methods that do not depend on re-tuning of their parameters can thus be used as a single reusable approach that does not need to be re-tuned to perform well for solving any given problem. The existing research relating to cross-domain search has focused on developing improved heuristic selection (generation) components of selection (generation) hyper-heuristics as such cross-domain search methods, and this is despite the suggestion that the move acceptance component of a selection hyper-heuristic having more effect on its performance than the heuristic selection. The focus of this work was therefore on the move acceptance method component as these have not been studied in-depth before in the context of cross-domain search with the aim of designing an improved move acceptance method to improve the cross-domain performance of optimisation methods, but without the need for an expert to intervene and re-tune its parameters when tasked with solving problems new and unknown as previously unseen during its training phase of development.

6.2 Summary of Work

The work presented in this thesis provides a thorough survey of move acceptance methods as used in single-point based hyper-heuristics, and classifies them based on a taxonomy that is proposed for classifying them based on the characteristics of their move acceptance methods. The cross-domain performance of existing move acceptance methods, with one being chosen from each classification from the taxonomy, used under a local search metaheuristic framework was compared using an empirical study, and the effects of parameter tuning versus the choice of the move acceptance method was explored. A novel move acceptance method (HAMSTA) was then proposed which has a cross-domain performance that improves over the existing methods despite HAMSTA using a single parameter configuration, requiring no expert intervention, for solving all COPs whereas the existing methods HAMSTA was compared to utilise different parameter configurations for each problem domain.

6.2.1 Local Search Metaheuristics and Cross-domain Search

This section of the thesis started with a discussion of the related scientific literature. After which, a taxonomy for classifying move acceptance methods based on their different natures was proposed and a survey of these algorithms used in local search methods was performed and classified based on the taxonomy. This was done since the existing taxonomies for local search metaheuristics do not go any further than that to say that they are single-point based single-objective heuristic search methods and no emphasis is made on the characteristics of their embedded move acceptance methods. The taxonomy was proposed with the aim of being able to classify different move acceptance methods, and to be able to observe the performance of different methods for solving different COPs. The outcome of the survey showed that even in the scope of studying single-point based single-objective move acceptance methods, there is a vast number of existing methods. The majority of these methods use an adaptive nature of the algorithmic parameter settings; however, there was no tendency for these methods to use a particular nature of the accept/reject decision component.

6.2.2 On the Cross-domain Performance of Move Acceptance Methods

The investigation into the cross-domain performance of move acceptance methods in local search metaheuristics in the context of cross-domain search has not been done before. This meant that when a researcher or practitioner was tasked with solving a *new* or *unknown* problem, they are left with a dilemma for choosing the best move acceptance method for solving that problem without previous experience, knowledge, or guidance. In this chapter, an empirical study comparing the performance of move acceptance methods, one from each of the possible classifications, was carried out comparing their performance across 9 characteristically different COPs under a single point-based search local search metaheuristic framework, representing the cross-domain search problem. The empirical study on the existing move acceptance methods suggests that if a practitioner is seeking guidelines for which move acceptance method to use for solving a *new*, *unknown*, or *existing* problem that is not covered in this study, then Simulated Annealing is recommended since it has the best performance over the cross-domain benchmark. If the problem to be solved however is *known* and covered in this study, and a simple single point-based stochastic local search metaheuristic framework is preferred, then the move acceptance method recommended for use is that which has the best per-domain performance, as stylised bold in Table 4.1, for that problem domain.

6.2.3 The History-based Adaptive Multi-Stage Threshold Accepting Algorithm

This chapter proposed a novel move acceptance method (HAMSTA) which aims to perform as-good-as the move acceptance methods which are tuned per-domain, but without itself requiring such expert intervention. The work in this chapter therefore contributes a single move acceptance method (HAMSTA) that is capable of solving most COPs to a high-quality without the need for expert intervention in the form of parameter tuning, and is shown to outperform the existing move acceptance methods that are re-tuned for each problem domain under a local search metaheuristic framework. In doing so, the decision-making dilemma faced by researchers and practitioners when solving a new or unknown problem can be eliminated from the decision-making process as; (1)

HAMSTA having the best cross-domain performance is proposed as the algorithm of choice for solving new and unknown problems, (2) HAMSTA outperforming the existing move acceptance methods from the study using only a single parameter configuration does not need to be re-tuned for it to perform well - this is despite the existing move acceptance methods being re-tuned for each domain, and (3) HAMSTA having the best cross-domain performance means that it reasonable to expect that, at this point in time, HAMSTA can outperform the move acceptance methods from this study for use with a local search metaheuristic framework. While it can be argued that higher-level algorithmic frameworks such as hyper-heuristics may perform better, using HAMSTA under a local search metaheuristic framework also has the advantage that a when a practitioner is tasked with solving a new or unknown problem, or a researcher wants a baseline solution to a problem, they not only gets to use an off-the-shelf algorithm that performs well and does not need to be re-tuned, but they are only required to design a simple perturbative move operator for the given problem; there is no need for them to design a portfolio of specialised heuristics which can be up-front time consuming and costly in itself.

6.3 Future Work

Cross-domain Search

The trend since research into solving the cross-domain search problem began in 2011 focuses on creating search methods that are able to solve real-world COPs well using a fixed computational budget of 10 nominal minutes - the time as defined by the Cross-domain Heuristic Search Challenge (CHeSC 2011) competition. One of the criticisms that cross-domain search receives is that these algorithms are designed to work well for a fixed time period. A future research direction for cross-domain search, which resolves this controversy of the existing problem definition, is that the cross-domain performance of a cross-domain search method should be evaluated over a range of computational budgets. Moreover, the aim of cross-domain search research to produce a high-level problem solver is of little use if the solution method requires expert intervention such as parameter tuning for it to perform well when tasked with solving multiple problems. Thus, a proposition is made to extend the current definition of cross-domain search to:

Proposed Definition: Cross-domain search is the high-level issue of designing a single general-purpose optimisation method which can be used to solve multiple characteristically different optimisation problems to a high quality given *any* pre-defined computational budget *and without the need for* expert intervention beyond their design and training phases, and irrespective of any additions or changes to the problems or computational budgets that they may solve.

For example, a related area of research concerns the anytime behaviour [179, 180] of optimisation algorithms. This has a similar goal of solving a problem to a high quality but where the computational budget is not known in advance; hence, these algorithms are designed to produce high-quality solutions irrespective of the computational budget. The outputs from those studies could therefore be used to design algorithms that can solve problems to a high quality given a range of pre-defined computational budgets but without the need for expert intervention to re-configure them.

Furthermore, the problems that are studied as part of the cross-domain search problem to date are only single objective optimisation problems and involve "traditional" COPs where the computational expense of their objective functions are not much of a concern. Multi-objective optimisation problems are those where the problem includes multiple conflicting and incommensurable objectives that need to maximised or minimised and requires specialised algorithms to solve them such as NSGA-II [181], SPEA2 [182] and IBEA [183]. Many-objective optimisation problems are multi-objective optimisation problems that contain four or more conflicting and incommensurable objectives [184] and the algorithms designed for solving multi-objective optimisation problems can very quickly become overwhelmed with the additional complexity. Refinements to, and additional algorithms, such as MOEA/D [185] and MOEA/DD [186] were therefore proposed for solving such problems and other extensive works were conducted [187] comparing the components of existing MO/MaO algorithms on MaOPs. Another area of research called inverse combinatorial optimisation, includes such problems where the cost of evaluating the objective value of a solution is significantly greater than the time taken by the move operators [188, 189, 190, 191]. Thus, combining COPs that are characteristically different in the nature of their objective functions into the set of benchmark problems could be explored such as the traditional and inverse COPs, and single/multi/many objective COPs. This makes sense from the point of view of generalising cross-domain search methods as general-purpose optimisation methods, but a mixture of these problem characteristics and objectives would pose a significant challenge for a single algorithm and possibly warrants a new research area of its own.

HAMSTA Beyond the Local Search Metaheuristic Framework

The work in this thesis focuses on a local search metaheuristic framework as provided in Algorithm 1, perturbing a solution and then making an accept/reject decision for the new solution. This was done in order to remove as many confounding factors as possible from the effects of the move acceptance methods on the cross-domain performance of an optimisation method. HAMSTA, as the move acceptance method, is shown to improve the cross-domain performance of a local search metaheuristic while making use of only a single parameter configuration for solving all problems, but other search frameworks exist that could benefit from using HAMSTA as its sole or one of its multiple move acceptance methods. While the work in this thesis compares move acceptance methods using a single-point based stochastic local search metaheuristic framework, the work that is presented is still of value to the wider community since this experimentation can be generalised and extended to other local search frameworks and their hybrids [192]. For example, an Iterated Local Search framework [193] perturbs and then applies hill climbing on an incumbent solution, can be investigated using various move acceptance methods. The concept of applying such acceptance criteria is discussed in [193], referring to *Better*, *RW*, and *LMSC* as such strategies.

As another example, Tabu Search, one of the very well-known approaches, that was not considered in this study due to differences in the way in which it operates compared to the local search metaheuristic framework as described in Chapter 1. Tabu Search [194, 195] also carries forward a single solution from one pass of the algorithm to the next as a single-point search method, however its neighbourhood structure is very different. In Tabu Search, the basic principle is to choose the best solution from the current solution's neighbourhood that is not prohibited by a tabu list which prevents the search from re-tracing its previous steps. Tabu Search therefore has to evaluate all neighbouring solutions as defined by its neighbourhood operator, and chooses the best one to proceed, $s_{i+1} \leftarrow bestAdmissible \in (s)$. In the local search metaheuristic framework explained prior on the other hand, a single neighbouring solution is chosen completely at random by performing a perturbative change to the solution-in-hand. There are studies which hybridise these single-point based search metaheuristics within the Tabu Search framework such as in [196] where Tabu Search is used to find a candidate solution from the solution neighbourhood, but where the acceptance of the candidates solution is determined by Simulated Annealing. Another trivial research direction would be to investigate the performance of different move acceptance methods studied in this thesis under the framework suggested by [196].

Abstracting an Iterative Multi-stage Search Framework from HAMSTA

Perhaps unsurprisingly, preliminary testing of the HAMSTA-based local search metaheuristic shows that it does not have as good of a cross-domain performance when compared to the state-of-the-art hyper-heuristics. In this work, the groundworks are laid for future research directions of move acceptance methods as components of heuristic optimisation methods. It is evident that even for a local search metaheuristic, which uses a single perturbative operator, more complex move acceptance methods than the existing ones are required for them to perform well under a cross-domain setting. They are most often designed to balance exploitation and exploration throughout the search process by controlling their algorithmic parameters. For example, the temperature in SA is reduced over time to favour exploitation in the latter stages compared to exploration in the earlier stages, GD maintains a threshold which is reduced over time to initially allow many worse moves but over time such worse moves are only accepted if their quality is satisfactory given the remaining computational budget of the search.

A general iterative multi-stage search framework can be extracted from the design of HAMSTA as shown in Figure 6.1. This framework is composed of three stages; exploitation, exploration, and a stage which balances these such that at first an exploitation stage is executed, followed by an iterative application of an exploration stage, and the stage that balances exploitation and exploration. HAMSTA uses IE, AM and HTA for each of these stages respectively.

An optimisation algorithm under the proposed iterative multi-stage search framework would start by initialising its internal parameters/adaptation mechanisms as shown by *InitialisationProcedure_1();*. Then, the exploration stage is invoked and ran until its termination criterion is met. The variables and mechanisms can then be set up for the subsequent stages, as shown by *InitilaisaitonProcedure_2();* and the first exploration stage can then ran. This exploration stage then continues until its termination criterion is met, and the subsequent stage is decided based upon whether the previous exploration stage was able to improve its initial solution or not. Subsequent to an improving exploitation stage, the framework then enters an iterative process of applying the balanced exploitation and exploration stage followed by the exploration stage. Upon the completion of the balanced exploitation and exploration stage, an adaptation mechanism should be employed to reconfigure the algorithmic parameters of each stage based on the search history as shown by *AdaptationProcedure();*. An algorithm under the search framework terminates at any point subject to the computational budget.

The difference that is meant between the exploitation and balanced move acceptance strategies

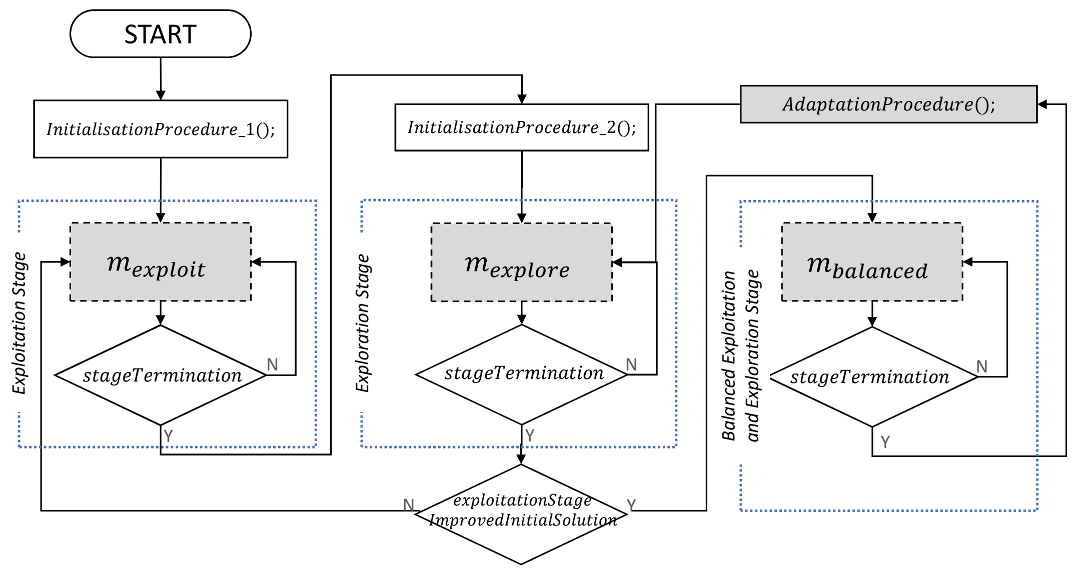

START
InitialisationProcedure_1();
InitialisationProcedure_2();
AdaptationProcedure();
Exploitation Stage
Exploration Stage
Balanced Exploitation and Exploration Stage
$m_{exploit}$
$m_{explore}$
$m_{balanced}$
stageTermination
N
Y
exploitationStage ImprovedInitialSolution

Figure 6.1: An iterative multi-stage search framework based on the strategies used by the HAMSTA move acceptance method where $m_{exploit}$ is a move acceptance method used to rapidly improve the solution-in-hand, $m_{explore}$ is a move acceptance method which allows an exploration of the search space, and $m_{balanced}$ is a move acceptance method which is designed to balance appropriately its exploitation and exploration abilities based on the local search landscape region.

is that the exploitation stage exploits the search neighbourhood by only accepting non-worsening moves to improve the solution-in-hand, whereas the balanced strategy exploits the search history to allow both non-worsening and worse moves in a balanced and controlled manner to improve the solution-in-hand over time.

The existing hyper-heuristics make use of iterated single-stage algorithms or sequential multi-stage algorithms for their move acceptance method, but the use of iterative multi-stage move acceptance methods has not been explored. HAMSTA, as a move acceptance method which employs an internal adaptive iterative multi-stage mechanism, improves over the existing move acceptance methods under a local search metaheuristic framework. Hyper-heuristics operate at a higher-level than metaheuristics and contain a set of low-level heuristics of various types such as: local search, mutation (perturbation), crossover, and ruin-recreate. Hyper-heuristics shift the focus away from the search space of solutions, and instead centre on the search space of the heuristics. If a simple heuristic search framework, i.e. local search metaheuristics, requires specially designed move acceptance methods for improving their cross-domain performance, then more advanced methods combining multiple neighbourhood operators such as hyper-heuristics should also benefit from the design of such move acceptance methods. Given the success of HAMSTA, it is reasonable to expect that some form of iterative multi-stage algorithm can improve the cross-domain performance of a heuristic-based algorithm further. Therefore, a future research direction should include the investigation of *iterative multi-stage move acceptance methods* under a hyper-heuristic framework to advance the current state-of-the-art cross-domain search methods - this has not been previously explored.

Further studies could include the investigation of the effects of using different move acceptance components under the iterative multi-stage framework where, for example, HTA could be replaced with other methods for the $m_{balanced}$ stage such as Simulated Annealing or Great Deluge, or even HAMSTA. Moreover, machine learning techniques could be embedded into the search framework such that the move acceptance method used for $m_{balanced}$ could be selected at each $m_{balanced}$ stage from a set of move acceptance methods. Furthermore, ensemble methods such as group decision-making could also be employed to mix multiple move acceptance methods during the same stage. The results of previous studies have shown the advantages of using ensembles such as group decision-making for combining the decisions of multiple move acceptance methods [197], and more recently they have been investigated for solving Bin Packing problems [198] where it was found that randomly constructed ensembles can outperform ensembles of high-performance methods. Using a similar strategy to mix multiple move acceptance methods for the "balanced" stage within the iterative multi-stage framework could also prove beneficial to its cross-domain performance, especially considering that an algorithm for cross-domain search acts on characteristically different COPs where some move acceptance methods may perform well, whereas others may not. Moreover, online learning strategies have been used under a hyper-heuristic framework [97] to decide the best heuristic to apply from a set of low-level heuristics. Similar online learning strategies could be employed to choose which move acceptance method to use during the exploitation-exploration stage of HAMSTA.

The Role of Move Acceptance for Cross-domain Search

The level of generality at which search methods can act have been raised through the use of hyper-heuristics in the past, allowing them to solve problems from different domains without modification. Realistically however, their effectiveness for solving problems well from different domains is not as good as they should be and remains in the focus of current research. Since the design of AdapHH in 2011 [199], only a few methods have been able to improve upon its cross-domain performance. This was either achieved using machine learning techniques for improving heuristic selection [94], or by using accidental complexity analysis which aims to optimise the existing design of AdapHH (GIHH) [200] by simplifying its design by removing unnecessary mechanisms built into the algorithm by the human expert. In this work, we have shown that even under a simple local search metaheuristic framework, the choice of move acceptance method can have a significant effect on the cross-domain performance of such search methods. The variation in performance of each of the move acceptance methods can even be seen across each of the problem domains despite performing parameter tuning for solving each problem. Considering these outcomes, a future research direction to improve the effectiveness of general purpose search methods should focus on improving the move acceptance components by designing them specifically for solving the cross-domain search problem - one of such methods is successfully presented in this work as HAMSTA; however, further ideas are presented below which could have the potential to further improve the performance of cross-domain search methods through new approaches for move acceptance.

One approach could involve the use of multi-stage algorithms which utilise different move acceptance methods throughout the search. This has the advantage of being able to mix such methods which are, with respect to our taxonomy, characteristically different. In theory, such search methods could then use search history to adapt the search process by changing between each of them as required to effectively solve the problem in hand. A multi-stage approach was under a hyper-heuristic framework in [94] employing two move acceptance strategies, greedy and threshold acceptance; however, at each stage the greedy strategy is invoked stochastically, employing threshold acceptance in the case that greedy is not chosen. There is a lack of the use of an exploration stage in and there is no explicit strategy to enforce an iterative application of its stages. HAMSTA on the other hand is an example of a move acceptance method under an iterative multi-stage search framework and internally uses three move acceptance method coming from each *non-stochastic basic* and *non-stochastic threshold* classifications and shows promising potential for further work under this framework.

Another possibility is to select a single move acceptance method, or use the decisions, from a set of move acceptance methods where the move acceptance methods should span the different natures of the accept/reject decision from the taxonomy presented in this work. Previous work has involved the mixture of move acceptance criteria. Tensor analysis, as an advanced machine learning technique, was used in [40] to associate a set of move operators with two move acceptance methods within a hyper-heuristic. Each move acceptance method was used in a phase based approach where each subsequent phase employed the opposite move acceptance method to the one currently used. Group decision-making was used in [197], and is a technique used to collaboratively arrive at a decision to accept or reject a move based on several independent move acceptance methods. Each move acceptance method within a group decision-making strategy can be assigned a weight to change the influence that each move acceptance method has in the overall decision. Furthermore, adaptation of these weights based on the nature of the accept/reject decision of the move acceptance methods can be used to increase the influence that each has based on the state of the search. These are just some of the ways in which move acceptance methods using different natures of accept/reject decisions can be used together under the same search method.

References

[1] D. Landa-Silva and J. H. Obit, "Great deluge with non-linear decay rate for solving course timetabling problems," in *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, vol. 1, pp. 8–11, Sept 2008.

[2] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[3] S. Petrovic, Y. Yang, and M. Dror, "Case-based selection of initialisation heuristics for metaheuristic examination timetabling," *Expert Systems with Applications*, vol. 33, no. 3, pp. 772–785, 2007.

[4] L. N. Ahmed, E. Özcan, and A. Kheiri, "Solving high school timetabling problems worldwide using selection hyper-heuristics," *Expert Systems with Applications*, vol. 42, pp. 5463–5471, Aug. 2015.

[5] C. D. Tarantilis, C. T. Kiranoudis, and V. S. Vassiliadis, "A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem," *European Journal of Operational Research*, vol. 152, no. 1, pp. 148–158, 2004.

[6] K. Sörensen and F. W. Glover, "Metaheuristics," in *Encyclopedia of Operations Research and Management Science*, pp. 960–970, Springer US, 2013.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1990.

[8] G. Ausiello, P. Crescenzi, and M. Protasi, "Approximate solution of {NP} optimization problems," *Theoretical Computer Science*, vol. 150, no. 1, pp. 1–55, 1995.

[9] Z. Drezner, P. M. Hahn, and É. D. Taillard, "Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods," *Annals of Operations Research*, vol. 139, pp. 65–94, Oct 2005.

[10] P. Hahn, Y.-R. Zhu, M. Guignard-Spielberg, W. Hightower, and M. Saltzman, "A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem," *INFORMS Journal on Computing*, vol. 24, pp. 202–209, 2012.

[11] W. Adams and L. Waddell, "Linear programming insights into solvable cases of the quadratic assignment problem," *Discrete Optimization*, vol. 14, pp. 46 – 60, 2014.

[12] Z. Beheshti and S. M. Shamsuddin, "A review of population-based meta-heuristic algorithm," *International Journal of Advances in Soft Computing and its Applications*, vol. 5, no. 1, 2013.

[13] M. Birattari, L. Paquete, T. Stützle, and K. Varrentrapp, "Classification of metaheuristics and design of experiments for the analysis of components," tech. rep., Intellektik Darmstadt University of Technology, Darmstadt, Germany, 2001.

[14] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, pp. 268–308, September 2003.

[15] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[16] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*, vol. 45 of *Operations Research/Computer Science Interfaces*. Springer Verlag, 2008.

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

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[19] G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *Journal of Computational Physics*, vol. 104, no. 1, pp. 86–92, 1993.

[20] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. K. Burke, *HyFlex: A Benchmark*

Framework for Cross-Domain Heuristic Search, pp. 136–147. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[21] C. Igel, *No Free Lunch Theorems: Limitations and Perspectives of Metaheuristics*, pp. 1–23. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[22] G. Ochoa and M. Hyde, "The cross-domain heuristic search challenge (chesc 2011)." [Online]. Available: `http://www.asap.cs.nott.ac.uk/external/chesc2011`, 2011. Accessed: 11-05-2019.

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

[24] J. McDermott, "When and why metaheuristics researchers can ignore "no free lunch" theorems," *SN Computer Science*, vol. 1, Jan 2020.

[25] A. Santini, S. Ropke, and L. M. Hvattum, "A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic," *Journal of Heuristics*, vol. 24, pp. 783–815, October 2018.

[26] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, pp. 3–23, Jan 2008.

[27] J. Ries, P. Beullens, and Y. Wang, *Instance-specific parameter tuning for meta-heuristics*, pp. 136–170. IGI Global, 9 2012.

[28] E. K. Burke and Y. Bykov, "A late acceptance strategy in hill-climbing for exam timetabling problems," in *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, (Montreal, Canada), p. Extended Abstract, 2008.

[29] B. Adenso-Diaz and M. Laguna, "Fine-tuning of algorithms using fractional experimental designs and local search," *Operations Research*, vol. 54, no. 1, pp. 99–114, 2006.

[30] C. Huang, Y. Li, and X. Yao, "A survey of automatic parameter tuning methods for metaheuristics," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 201–216, 2020.

[31] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.

[32] L. Lindawati, *Generic Instance-Specific Automated Parameter Tuning Framework*. PhD thesis, Singapore Management University, Singapore, 2014. Available: `https://ink.library.smu.edu.sg/etd_coll/100`.

[33] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43 – 58, 2016.

[34] A. J. Benavides and M. Ritt, "Iterated local search heuristics for minimizing total completion time in permutation and non-permutation flow shops," in *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'15, pp. 34–41, AAAI Press, 2015.

[35] B. L. Golden and C. C. Skiscim, "Using simulated annealing to solve routing and location problems," *Naval Research Logistics Quarterly*, vol. 33, no. 2, pp. 261–279, 1986.

[36] F. Li, B. Golden, and E. Wasil, "A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem," *Computers & Operations Research*, vol. 34, no. 9, pp. 2734 – 2742, 2007.

[37] P. I. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, (London, UK, UK), pp. 176–190, Springer-Verlag, 2001.

[38] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: Revisited," in *Handbook of Metaheuristics*, pp. 453–477, Springer, 2019.

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

[40] S. Asta and E. Özcan, "A tensor-based selection hyper-heuristic for cross-domain heuristic search," *Information Sciences*, vol. 299, pp. 412–432, 2015.

[41] S. S. Choong, L.-P. Wong, and C. P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," *Information Sciences*, vol. 436-437, pp. 89 – 107, 2018.

[42] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Applied Soft Computing*, vol. 11, no. 6, pp. 4135–4151, 2011.

[43] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, pp. 239–287, Jun 2009.

[44] E. Hopper and B. C. H. Turton, "A review of the application of meta-heuristic algorithms to 2d strip packing problems," *Artificial Intelligence Review*, vol. 16, no. 4, pp. 257–300, 2001.

[45] R. Lewis, "A survey of metaheuristic-based techniques for university timetabling problems," *OR Spectrum*, vol. 30, no. 1, pp. 167–190, 2008.

[46] P. Cowling, G. Kendall, and Limin Han, "An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 2, pp. 1185–1190 vol.2, 2002.

[47] X. Wu, P. Consoli, L. Minku, G. Ochoa, and X. Yao, "An evolutionary hyper-heuristic for the software project scheduling problem," in *Parallel Problem Solving from Nature – PPSN XIV* (J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, eds.), (Cham), pp. 37–47, Springer International Publishing, 2016.

[48] E. Özcan, J. H. Drake, C. Altıntaş, and S. Asta, "A self-adaptive multimeme memetic algorithm co-evolving utility scores to control genetic operators and their parameter settings," *Applied Soft Computing*, vol. 49, pp. 81 – 93, 2016.

[49] D. B. Gümüş, E. Özcan, and J. Atkin, "An investigation of tuning a memetic algorithm for cross-domain search," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 135–142, July 2016.

[50] R. J. Marshall, M. Johnston, and M. Zhang, "Hyper-heuristic operator selection and acceptance criteria," in *Evolutionary Computation in Combinatorial Optimization* (G. Ochoa and F. Chicano, eds.), (Cham), pp. 99–113, Springer International Publishing, 2015.

[51] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 3, pp. 124–141, Jul 1999.

[52] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[53] B. Bilgin, E. Özcan, and E. E. Korkmaz, “An experimental study on hyper-heuristics and exam timetabling,” in *Practice and Theory of Automated Timetabling VI*, (Berlin, Heidelberg), pp. 394–412, Springer Berlin Heidelberg, 2007.

[54] I. Maden, Ş. Uyar, and E. Özcan, “Landscape analysis of simple perturbative hyper-heuristics,” in *Mendel 2009: 15th International Conference on Soft Computing*, pp. 16–22, 2009.

[55] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, “Grammatical evolution hyper-heuristic for combinatorial optimization problems,” *Evolutionary Computation, IEEE Transactions on*, vol. 17, pp. 840–861, 2013.

[56] G. Kendall and M. Mohamad, “Channel assignment optimisation using a hyper-heuristic,” in *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, vol. 2, pp. 791–796, Dec 2004.

[57] M. Ayob and G. Kendall, “A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine,” in *Placement Machine, INTECH’03 Thailand*, pp. 132–141, 2003.

[58] T. Wauters, W. Vancroonenburg, and G. Vanden Berghe, “A guide-and-observe hyper-heuristic approach to the eternity ii puzzle,” *Journal of Mathematical Modelling and Algorithms*, vol. 11, no. 3, pp. 217–233, 2012.

[59] B. Kiraz and H. Topcuoglu, “Hyper-heuristic approaches for the dynamic generalized assignment problem,” in *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pp. 1487–1492, Nov 2010.

[60] A. Kheiri and E. Keedwell, “A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems,” *Evolutionary Computation*, vol. 25, no. 3, pp. 473–501, 2017. PMID: 27258841.

[61] S. M. Pour, J. Drake, and E. Burke, “A choice function hyper-heuristic framework for the allocation of maintenance tasks in danish railways,” *Computers and Operations Research*, vol. 93, pp. 15–26, 2018.

[62] 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.

[63] E. Soubeiga, *Development and application of hyperheuristics to personnel scheduling.* PhD thesis, University of Nottingham, 2003.

[64] Z. Ren, H. Jiang, J. Xuan, and Z. Luo, "Ant based hyper heuristics with space reduction: A case study of the p-median problem," in *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, PPSN'10, (Berlin, Heidelberg), pp. 546–555, Springer-Verlag, 2010.

[65] D. Ouelhadj and S. Petrovic, "A cooperative distributed hyper-heuristic framework for scheduling," in *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pp. 2560–2565, Oct 2008.

[66] E. K. Burke, G. Kendall, D. Landa Silva, R. O'Brien, and E. Soubeiga, "An ant algorithm hyperheuristic for the project presentation scheduling problem," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, pp. 2263–2270 Vol. 3, Sept 2005.

[67] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation," in *Applications of Evolutionary Computing* (S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, eds.), (Berlin, Heidelberg), pp. 1–10, Springer Berlin Heidelberg, 2002.

[68] P. I. Cowling, G. Kendall, and E. Soubeiga, "A parameter-free hyperheuristic for scheduling a sales summit," in *Proceedings of the 4th Metaheuristic International Conference, MIC 2001*, pp. 127–131, 2001.

[69] R. Bai and G. Kendall, "An investigation of automated planograms using a simulated annealing based hyper-heuristics," in *Progress as Real Problem Solver - (Operations Research/Computer Science Interface Series, Vol.32*, pp. 87–108, Springer, 2005.

[70] A. Berberoğlu and A. c. Uyar, "Experimental comparison of selection hyper-heuristics for the short-term electrical power generation scheduling problem," in *Applications of Evolutionary Computation*, vol. 6625 of *Lecture Notes in Computer Science*, pp. 444–453, Springer Berlin Heidelberg, 2011.

[71] J. Gibbs, G. Kendall, and E. Özcan, "Scheduling english football fixtures over the holiday period using hyper-heuristics," in *Parallel Problem Solving from Nature, PPSN XI* (R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, eds.), vol. 6238 of *Lecture Notes in Computer Science*, pp. 496–505, Springer Berlin Heidelberg, 2010.

[72] L. Di Gaspero and T. Urli, "Evaluation of a family of reinforcement learning cross-domain optimization heuristics," in *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pp. 384–389, Springer Berlin Heidelberg, 2012.

[73] S. Adriaensen, T. Brys, and A. Nowé, "Fair-share ils: A simple state-of-the-art iterated local search hyperheuristic," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, (New York, NY, USA), pp. 1303–1310, ACM, 2014.

[74] S. Adriaensen, T. Brys, and A. Nowé, "Designing reusable metaheuristic methods: A semi-automated approach," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2969–2976, July 2014.

[75] J. H. Drake, E. Özcan, and E. K. Burke, "A modified choice function hyper-heuristic controlling unary and binary operators," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3389–3396, May 2015.

[76] A. Kheiri and E. Keedwell, "A sequence-based selection hyper-heuristic utilising a hidden markov model," in *GECCO 2015 - Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, pp. 417–424, Association for Computing Machinery, Inc, 7 2015.

[77] 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, no. 1, pp. 177 – 192, 2007.

[78] E. K. Burke, M. Gendreau, G. Ochoa, and J. D. Walker, "Adaptive iterated local search for cross-domain optimisation," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, (New York, NY, USA), pp. 1987–1994, ACM, 2011.

[79] J. H. Drake, E. Özcan, and E. K. Burke, "Controlling crossover in a selection hyper-heuristic framework," *School of Computer Science, University of Nottingham, Tech. Rep. No. NOTTCS-TR-SUB-1104181638-4244*, 2011.

[80] J. H. Drake, E. Özcan, and E. K. Burke, "A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem," *Evolutionary Computation*, vol. 24, no. 1, pp. 113–141, 2016. PMID: 25635698.

[81] B. Bilgin, P. Demeester, M. Mısır, W. Vancroonenburg, and G. Vanden Berghe, "One hyper-heuristic approach to two timetabling problems in health care," *Journal of Heuristics*, vol. 18, no. 3, pp. 401–434, 2012.

[82] A. Berberoğlu and A. c. Uyar, “A hyper-heuristic approach for the unit commitment problem,” in *Applications of Evolutionary Computation*, vol. 6025 of *Lecture Notes in Computer Science*, pp. 121–130, Springer Berlin Heidelberg, 2010.

[83] E. Özcan, Ş. E. Uyar, and E. K. Burke, “A greedy hyper-heuristic in dynamic environments,” in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO ’09, (New York, NY, USA), pp. 2201–2204, ACM, 2009.

[84] P. Demeester, B. Bilgin, P. De Causmaecker, and G. Vanden Berghe, “A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice,” *Journal of Scheduling*, vol. 15, no. 1, pp. 83–103, 2012.

[85] M. Mısır, K. Verbeeck, P. De Causmaecker, and G. V. Berghe, “Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem,” in *IEEE Congress on Evolutionary Computation*, pp. 1–8, July 2010.

[86] M. Mısır, P. Smet, K. Verbeeck, and G. Vanden Berghe, “Security personnel routing and rostering: a hyper-heuristic approach,” in *the 3rd International Conference on Applied Operational Research (ICAOR’11)*, vol. 3, pp. 193–205, 2011.

[87] M. Mısır, W. Vancroonenburg, K. Verbeeck, and G. V. Berghe, “A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete,” in *Proceedings of the Metaheuristics International Conference (MIC 2011)*, pp. 289–298, 2011.

[88] M. Mısır, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe, “A new hyper-heuristic as a general problem solver: an implementation in hyflex,” *Journal of Scheduling*, vol. 16, pp. 291–311, Jun 2013.

[89] E. K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J. Vázquez-Rodríguez, and M. Gendreau, “Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms,” in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp. 1–8, 2010.

[90] E. Özcan and A. Kheiri, “A hyper-heuristic based on random gradient, greedy and dominance,” in *Computer and Information Sciences II* (E. Gelenbe, R. Lent, and G. Sakellari, eds.), pp. 557–563, Springer London, 2012.

[91] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, no. 1, pp. 161–175, 1990.

[92] Y. Bykov and S. Petrovic, "A step counting hill climbing algorithm applied to university examination timetabling," *Journal of Scheduling*, vol. 19, pp. 479–492, Aug 2016.

[93] M. Mısır, T. Wauters, K. Verbeeck, and G. Vanden Berghe, "A new learning hyper-heuristic for the traveling tournament problem," in *Proceedings of the 8th Metaheuristic International Conference (MIC09)*, 2009.

[94] A. Kheiri and E. Özcan, "An iterated multi-stage selection hyper-heuristic," *European Journal of Operational Research*, vol. 250, no. 1, pp. 77–90, 2016.

[95] M. Sinclair, "Comparison of the performance of modern heuristics for combinatorial optimization on real data," *Computers & Operations Research*, vol. 20, no. 7, pp. 687–695, 1993.

[96] G. Kendall and M. Mohamad, "Channel assignment in cellular communication using a great deluge hyper-heuristic," in *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, vol. 2, pp. 769–773 vol.2, Nov 2004.

[97] E. Özcan, M. Mısır, G. Ochoa, and E. K. Burke, "A reinforcement learning - great-deluge hyper-heuristic for examination timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, 2010.

[98] S. M. S. Bhanu and N. P. Gopalan, "A hyper-heuristic approach for efficient resource scheduling in grid," *International Journal of Computers Communications & Control*, vol. 3, pp. 249–258, 2008.

[99] E. K. Burke and Y. Bykov, "Solving exam timetabling problems with the flex-deluge algorithm," in *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pp. 370–372, 2006.

[100] E. S. Sin and N. S. M. Kham, "Hyper heuristic based on great deluge and its variants for exam timetabling problem," *International Journal of Artificial Intelligence & Applications (IJAIA)*, vol. 3, no. 1, pp. 149–162, 2012.

[101] J. H. Obit, D. Landa-Silva, M. Sevaux, and D. Ouelhadj, "Non-linear great deluge with reinforcement learning for university course timetabling," in *Metaheuristics: Intelligent Decision*

Making (M. Caserta and S. Voss, eds.), no. 50 in Operations Research/Computer Science Interfaces Series, New York: Springer, 2011.

[102] B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and S. Abdullah, "An extended great deluge approach to the examination timetabling problem," in *4th Multidisciplinary International Conference on Scheduling: Theory and Applications, 2009*, pp. 424–434, 2009.

[103] E. S. Sin, "Reinforcement learning with egd based hyper heuristic system for exam timetabling problem," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pp. 462–466, Sept 2011.

[104] P. McMullan, "An extended implementation of the great deluge algorithm for course timetabling," in *Computational Science – ICCS 2007* (Y. Shi, G. D. van Albada, J. Dongarra, and P. M. Sloot, eds.), vol. 4487 of *Lecture Notes in Computer Science*, pp. 538–545, Springer Berlin Heidelberg, 2007.

[105] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[106] A. S. Ferreira, R. A. Gonçalves, and A. T. R. Pozo, "A multi-armed bandit hyper-heuristic," in *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 13–18, Nov 2015.

[107] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *Evolutionary Computation, IEEE Transactions on*, vol. 19, pp. 309–325, June 2015.

[108] E. K. Burke, G. Kendall, M. Mısır, and E. Özcan, "Monte carlo hyper-heuristics for examination timetabling," *Annals of Operations Research*, vol. 196, no. 1, pp. 73–90, 2012.

[109] R. Bai, E. K. Burke, and G. Kendall, "Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation," *Journal of the Operational Research Society*, vol. 59, pp. 1387–1397, 2007.

[110] B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, G. Vanden Berghe, and T. Wauters, "A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition," in *The 8th international conference on the practice and theory of automated timetabling (PATAT'10)–the nurse rostering competition*, PATAT'10, 2010.

[111] M. Kalender, A. Kheiri, E. Ozcan, and E. K. Burke, “A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem,” in *Computational Intelligence (UKCI), 2012 12th UK Workshop on*, pp. 1–8, Sept 2012.

[112] M. Kalender, A. Kheiri, E. Özcan, and E. K. Burke, “A greedy gradient-simulated annealing selection hyper-heuristic,” *Soft Computing*, vol. 17, no. 12, pp. 2279–2292, 2013.

[113] D. Connolly, “General purpose simulated annealing,” *The Journal of the Operational Research Society*, vol. 43, no. 5, pp. 495–505, 1992.

[114] L. Ingber, “Very fast simulated re-annealing,” *Mathematical and Computer Modelling*, vol. 12, no. 8, pp. 967–973, 1989.

[115] R. Bai, J. Błażewicz, E. K. Burke, G. Kendall, and B. McCollum, “A simulated annealing hyper-heuristic methodology for flexible decision support,” *4OR*, vol. 10, no. 1, pp. 43–66, 2012.

[116] H. Jiang, S. Zhang, J. Xuan, and Y. Wu, “Frequency distribution based hyper-heuristic for the bin-packing problem,” in *Evolutionary Computation in Combinatorial Optimization* (P. Merz and J.-K. Hao, eds.), vol. 6622 of *Lecture Notes in Computer Science*, pp. 118–129, Springer Berlin Heidelberg, 2011.

[117] K. A. Dowsland, E. Soubeiga, and E. K. Burke, “A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation,” *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.

[118] A. Anagnostopoulos, L. Michel, P. V. Hentenryck, and Y. Vergados, “A simulated annealing approach to the traveling tournament problem,” *Journal of Scheduling*, vol. 9, pp. 177–193, Apr. 2006.

[119] E. Özcan, Y. Bykov, M. Birben, and E. Burke, “Examination timetabling using late acceptance hyper-heuristics,” in *Evolutionary Computation, 2009. CEC ’09. IEEE Congress on*, pp. 997–1004, May 2009.

[120] M. Mısır, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe, *An Intelligent Hyper-Heuristic Framework for CHeSC 2011*, pp. 461–466. Springer Berlin Heidelberg, 2012.

[121] R. Sridhar, M. Chandrasekaran, P. Sriramya, and S. Raja, “A review on application of 1d, 2d and 3d bin packing techniquies,” *Journal of Advanced Research in Dynamical and Control Systems*, vol. 2017, no. Special Issue 4, pp. 165–169, 2017.

[122] G. Scheithauer, "One-dimensional bin packing," *International Series in Operations Research and Management Science*, vol. 263, pp. 47–72, 2018.

[123] V. Fernandez-Viagas, R. Ruiz, and J. Framinan, "A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation," *European Journal of Operational Research*, vol. 257, no. 3, pp. 707–721, 2017.

[124] V. Fernandez-Viagas, J. Valente, and J. Framinan, "Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness," *Expert Systems with Applications*, vol. 94, pp. 58–69, 2018.

[125] A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *European Journal of Operational Research*, vol. 153, no. 1, pp. 3–27, 2004.

[126] S. Asta, E. Özcan, and T. Curtois, "A tensor based hyper-heuristic for nurse rostering," *Knowledge-Based Systems*, vol. 98, pp. 185–199, 2016.

[127] H. Santos, T. Toffolo, R. Gomes, and S. Ribas, "Integer programming techniques for the nurse rostering problem," *Annals of Operations Research*, vol. 239, no. 1, pp. 225–251, 2016.

[128] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided maxsat solving: A survey and assessment," *Constraints*, vol. 18, no. 4, pp. 478–534, 2013.

[129] C. Ansótegui, J. Gabàs, Y. Malitsky, and M. Sellmann, "Maxsat by improved instance-specific algorithm configuration," *Artificial Intelligence*, vol. 235, pp. 26–39, 2016.

[130] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, vol. 40. Princeton University Press, 2011.

[131] J. Escario, J. Jimenez, and J. Giron-Sierra, "Ant colony extended: Experiments on the travelling salesman problem," *Expert Systems with Applications*, vol. 42, no. 1, pp. 390–410, 2015.

[132] A. El-Shamir Ezugwu, A. Adewumi, and M. Frîncu, "Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem," *Expert Systems with Applications*, vol. 77, pp. 189–210, 2017.

[133] C. Lin, K. Choy, G. Ho, S. Chung, and H. Lam, "Survey of green vehicle routing problem: Past and future trends," *Expert Systems with Applications*, vol. 41, no. 4 PART 1, pp. 1118–1138, 2014.

[134] J. Montoya-Torres, J. López Franco, S. Nieto Isaza, H. Felizzola Jiménez, and N. Herazo-Padilla, "A literature review on the vehicle routing problem with multiple depots," *Computers and Industrial Engineering*, vol. 79, pp. 115–129, 2015.

[135] S. Adriaensen, G. Ochoa, and A. Nowé, "A benchmark set extension and comparative study for the hyflex framework," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pp. 784–791, IEEE, 2015.

[136] A. Fréville, "The multidimensional 0-1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004.

[137] Y. Zhou, X. Chen, and G. Zhou, "An improved monkey algorithm for a 0-1 knapsack problem," *Applied Soft Computing Journal*, vol. 38, pp. 817–830, 2016.

[138] T. Lim, M. Al-Betar, and A. Khader, "Taming the 0/1 knapsack problem with monogamous pairs genetic algorithm," *Expert Systems with Applications*, vol. 54, pp. 241–250, 2016.

[139] M. Etscheid and H. Röglin, "Smoothed analysis of local search for the maximum-cut problem," pp. 882–889, 2014.

[140] W. Ben-Ameur, A. R. Mahjoub, and J. Neto, "The maximum cut problem," in *Paradigms of Combinatorial Optimization*, ch. 6, pp. 131–172, John Wiley & Sons, Ltd, 2014.

[141] Y. Zhou, X. Lai, and K. Li, "Approximation and parameterized runtime analysis of evolutionary algorithms for the maximum cut problem," *IEEE Transactions on Cybernetics*, vol. 45, no. 8, pp. 1491–1498, 2015.

[142] T. Dokeroglu and A. Cosar, "A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem," *Engineering Applications of Artificial Intelligence*, vol. 52, pp. 10–25, 2016.

[143] F. Hafiz and A. Abdennour, "Particle swarm algorithm variants for the quadratic assignment problems - a probabilistic learning approach," *Expert Systems with Applications*, vol. 44, pp. 413–431, 2016.

[144] M. Hyde, G. Ochoa, T. Curtois, and J. A. Vazquez-Rodriguez, "A hyflex module for the one dimensional bin packing problem," tech. rep., School of Computer Science, University of Nottingham, 2010.

[145] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 3, pp. 299–325, 1974.

[146] ESICUP, "European special interest group on cutting and packing benchmark data sets.," 2011.

[147] M. R. Hyde, "One dimensional packing benchmark data sets," 2011.

[148] J. A. Vazquez-Rodriguez, G. Ochoa, T. Curtois, and M. Hyde, "A hyflex module for the permutation flow shop problem," tech. rep., School of Computer Science, University of Nottingham, 2010.

[149] M. Nawaz, E. Emory Enscore Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

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

[151] T. Curtois, G. Ochoa, M. Hyde, and J. A. Vazquez-Rodriguez, "A hyflex module for the personnel scheduling problem," tech. rep., School of Computer Science, University of Nottingham, 2010.

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

[153] T. Curtois, "Staff rostering benchmark data sets," 2009.

[154] M. Hyde, G. Ochoa, T. Curtois, and J. A. Vazquez-Rodriguez, "A hyflex module for the maximum satisfiability (max-sat) problem," tech. rep., School of Computer Science, University of Nottingham, 2010.

[155] CRIL, "Sat competition 2009 benchmark data sets.," 2009.

[156] CRIL, "Sat competition 2007 benchmark data sets," 2007.

[157] J. Argelich, C.-M. Li, F. Manya, and J. Planes, "Maxsat evalulation 2009 benchmark data sets," 2009.

[158] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968.

[159] G. Reinelt, "Tsplib, a library of sample instances for the tsp.," 2008.

[160] J. Walker, G. Ochoa, M. Gendreau, and E. K. Burke, "A vehicle routing domain for the hyflex hyper-heuristics framework," in *Proceedings of Learning and Intelligent Optimization (LION 2012)*, vol. 7219, pp. 265–276, 2012.

[161] J. D. Walker, G. Ochoa, M. Gendreau, and E. K. Burke, "Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework," in *Learning and Intelligent Optimization*, (Berlin, Heidelberg), pp. 265–276, Springer Berlin Heidelberg, 2012.

[162] SINTEF, "Vrptw benchmark problems, on the sintef transport optimisation portal," 2011.

[163] S. Adriaensen and G. Ochoa, "A hyflex module for the 0-1 knapsack problem," March 2015.

[164] D. Pisinger, "A more advanced generator for 0-1 knapsack problems," 2015.

[165] S. Adriaensen and G. Ochoa, "A hyflex module for the max cut problem," March 2015.

[166] DIMACS, "7th dimacs implementation challenge," 2015.

[167] G. Rinaldi, "Rudy graph generator," 2015.

[168] S. Adriaensen and G. Ochoa, "A hyflex module for the quadratic assignment problem," March 2015.

[169] R. E. Burkard, S. E. Karisch, and F. Rendl, "Qaplib - a quadratic assignment problem library," *Journal of Global Optimization*, vol. 10, no. 4, pp. 391–403, 1997.

[170] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization," *Journal of Heuristics*, vol. 15, p. 617, May 2008.

[171] R. K. Roy, *A primer on the Taguchi method.* Society of Manufacturing Engineers, 2010.

[172] H. Wang, Q. Geng, and Z. Qiao, "Parameter tuning of particle swarm optimization by using taguchi method and its application to motor design," in *2014 4th IEEE International Conference on Information Science and Technology*, pp. 722–726, April 2014.

[173] M. Lundy and A. Mees, "Convergence of an annealing algorithm," *Mathematical Programming*, vol. 34, no. 1, pp. 111–124, 1986.

[174] W. G. Jackson, E. Özcan, and R. I. John, "Tuning a simulated annealing metaheuristic for cross-domain search," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1055–1062, June 2017.

[175] D. Abramson, M. Krishnamoorthy, and H. Dang, "Simulated annealing cooling schedules for the school timetabling problem," *Asia-Pacific Journal of Operational Research*, pp. 1–22, 1999.

[176] K. Sim, "Ksats-hh: A simulated annealing hyper-heuristic with reinforcement learning and tabu-search," *Entry in the Cross-domain Heuristic Search Challenge available from http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html, June*, 2011.

[177] E. K. Burke and Y. Bykov, "The late acceptance hill-climbing heuristic," *European Journal of Operational Research*, vol. 258, no. 1, pp. 70 – 78, 2017.

[178] D. Meignan, "An evolutionary programming hyper-heuristic with co-evolution for chesc11," in *The 53rd Annual Conference of the UK Operational Research Society (OR53)*, vol. 3, 2011.

[179] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI Magazine*, vol. 17, no. 3, pp. 73–83, 1996.

[180] M. L.-I. nez and T. Stützle, "Automatically improving the anytime behaviour of optimisation algorithms," *European Journal of Operational Research*, vol. 235, no. 3, pp. 569 – 582, 2014.

[181] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, April 2002.

[182] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[183] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *International Conference on Parallel Problem Solving from Nature*, pp. 832–842, Springer, 2004.

[184] R. C. Purshouse and P. J. Fleming, "Evolutionary many-objective optimisation: an exploratory analysis," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 3, pp. 2066–2073 Vol.3, Dec 2003.

[185] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 712–731, Dec 2007.

[186] K. Li, K. Deb, Q. Zhang, and S. Kwong, "An evolutionary many-objective optimization algorithm based on dominance and decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 19, pp. 694–716, Oct 2015.

[187] L. Martí, E. Segredo, N. Sánchez-Pi, and E. Hart, "Selection methods and diversity preservation in many-objective evolutionary algorithms," *Data Technologies and Applications*, vol. 52, no. 4, pp. 502–519, 2018.

[188] D. Marc and M. Jérôme, *An Introduction to Inverse Combinatorial Problems*, ch. 17, pp. 547–586. Wiley-Blackwell, 2014.

[189] C. Heuberger, "Inverse combinatorial optimization: A survey on problems, methods, and results," *Journal of Combinatorial Optimization*, vol. 8, pp. 329–361, Sep 2004.

[190] E. D. Nino-Ruiz, C. Ardila, and R. Capacho, "Local search methods for the solution of implicit inverse problems," *Soft Computing*, pp. 1–14, 2017.

[191] S. Malcolm and M. Klaus, "Monte carlo methods in geophysical inverse problems," *Reviews of Geophysics*, vol. 40, no. 3, pp. 3–1–3–29, 2002.

[192] E.-G. Talbi, *A Unified Taxonomy of Hybrid Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning*, pp. 3–76. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[193] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," *International Series in Operations Research and Management Science*, vol. 57, pp. 321–354, 01 2003.

[194] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533 – 549, 1986. Applications of Integer Programming.

[195] F. Glover and M. Laguna, *Tabu search: effective strategies for hard problems in analytics and computational science*, vol. 21. 2013.

[196] I. H. Osman and N. Christofides, "Capacitated clustering problems by hybrid simulated annealing and tabu search," *International Transactions in Operational Research*, vol. 1, no. 3, pp. 317 – 336, 1994.

[197] A. Kheiri, M. Mısır, and E. Özcan, *Ensemble Move Acceptance in Selection Hyper-heuristics*, pp. 21–29. Springer International Publishing, 2016.

[198] E. Hart and K. Sim, "On constructing ensembles for combinatorial optimisation," *Evolutionary Computation*, vol. 26, pp. 67–87, 2018. REF compliant 29/11/2016 LG.

[199] M. Mısır, P. De Causmaecker, G. Vanden Berghe, and K. Verbeeck, "An adaptive hyper-heuristic for chesc 2011," 2011.

[200] S. Adriaensen and A. Nowé, "Case study: An analysis of accidental complexity in a state-of-the-art hyper-heuristic for hyflex," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1485–1492, July 2016.

Appendices

A Online Supplementary Data

- The complete set of results for comparing move acceptance methods (re-tuned per domain) under a local search metaheuristic framework as used in Chapter 4.2 — [`http://dx.doi.org/10.13140/RG.2.2.18709.37605`].
- The complete set of results for the cross-domain comparison of parameter tuning approaches vs choice of the move acceptance method as used in Chapter 4.3 — [`http://dx.doi.org/10.13140/RG.2.2.14307.99364`].
- The complete set of results for comparing HAMSTA tuned cross-domain to the existing move acceptance methods as used in Chapter 5 — [`https://www.doi.org/10.13140/RG.2.2.28146.56004/1`].

B Results and Pairwise Comparisons of HAMSTA to the Benchmark Move Acceptance Methods

The following tables report the best, mean average, and standard deviation of results of HAMSTA tuned cross-domain and each of the benchmark move acceptance methods tuned cross-domain and per-domain where feasible. Results of pairwise tests are included for completeness in the form of a Wilcoxon Signed Rank (non-parametric and paired) with CI = 95%. The complete set of results are available online here: `https://www.doi.org/10.13140/RG.2.2.28146.56004/1`.

Table 1: Results and pairwise comparisons using Wilcoxon Signed Rank test (CI = 95%) for each problem instance of HAMSTA (cross-domain tuned) to the cross-domain tuned move acceptance methods where $<$ ($>$) denotes that HAMSTA performs better (worse) than the associated move acceptance method, $\leq$ ($\geq$) denotes that HAMSTA does not perform statistically significantly different from the associated method, but performs on (mean) average better (worse). $\equiv$ denotes that HAMSTA and the associated algorithm have equivalent performance.

(a) Results of HAMSTA and pairwise comparison to IE.

		HAMSTA			IE			
P.	Inst.	Best	Mean	std.	Best	Mean	Std.	vs.
BP	1	3.394×10^{-03}	5.258×10^{-03}	1.719×10^{-03}	3.392×10^{-03}	4.450×10^{-03}	1.526×10^{-03}	$>$
	7	5.781×10^{-03}	8.543×10^{-03}	9.380×10^{-04}	5.720×10^{-03}	7.994×10^{-03}	7.724×10^{-04}	$>$
	9	0.0144	0.0157	7.918×10^{-04}	0.0123	0.0137	6.505×10^{-04}	$>$
	10	0.1086	0.1091	6.849×10^{-04}	0.1085	0.1088	5.215×10^{-04}	$>$
	11	0.0263	0.0276	9.894×10^{-04}	0.0225	0.0244	9.532×10^{-04}	$>$
FS	1	6250.00	6280.06	17.15	6236.00	6279.00	23.81	$\geq$
	3	6310.00	6352.65	21.37	6319.00	6345.90	19.50	$\geq$
	8	26754.00	26820.19	33.91	26750.00	26804.32	33.62	$>$
	10	11352.00	11414.68	37.48	11320.00	11385.68	32.93	$>$
	11	26591.00	26649.32	43.30	26555.00	26638.87	33.82	$\geq$
PS	5	38.00	49.68	6.00	688.00	1223.42	364.82	$<$
	8	43562.00	48469.23	2050.65	58640.00	60041.94	885.82	$<$
	9	63518.00	83842.81	6045.60	93131.00	99145.97	3507.66	$<$
	10	1520.00	1653.94	65.48	2165.00	3036.16	713.04	$<$
	11	415.00	513.74	55.94	1805.00	4550.13	1403.64	$<$
SAT	3	7.00	13.61	3.87	21.00	30.10	5.64	$<$
	4	4.00	13.94	6.10	17.00	40.42	7.46	$<$
	5	7.00	23.81	12.54	44.00	54.94	5.88	$<$
	10	6.00	11.06	2.98	22.00	27.65	3.89	$<$
	11	7.00	10.13	1.36	9.00	12.81	2.04	$<$
TSP	0	56845.65	60849.40	2773.94	58241.52	61397.45	2654.51	$<$
	2	7806.49	8013.44	136.49	7806.49	8015.58	137.54	$\leq$
	6	57843.55	59683.69	1073.44	57783.36	59660.40	1080.48	$>$
	7	74894.97	76674.23	697.67	74837.82	76636.78	703.42	$>$
	8	24787330.82	25046171.88	142593.00	24766167.28	25027169.31	142422.07	$>$
VRP	1	20672.57	20868.33	382.43	22763.06	24089.92	877.80	$<$
	2	12283.32	13382.69	505.43	14552.51	16155.79	652.04	$<$
	5	195424.25	205064.48	4191.12	195424.25	205237.19	4229.97	$\leq$
	6	107792.49	112929.04	3195.38	107791.35	113050.05	3229.83	$<$
	9	161371.90	165137.43	2594.57	161714.36	165420.08	2437.13	$<$
KP	0	1.034×10^{-05}	1.034×10^{-05}	5.166×10^{-21}	1.034×10^{-05}	1.034×10^{-05}	5.166×10^{-21}	$\equiv$
	1	2.673×10^{-06}	3.206×10^{-06}	1.648×10^{-07}	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$
	3	4.163×10^{-06}	4.315×10^{-06}	7.118×10^{-08}	5.202×10^{-06}	5.202×10^{-06}	8.610×10^{-22}	$<$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.139×10^{-06}	1.153×10^{-06}	6.587×10^{-09}	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$
MAC	0	2.416×10^{-08}	2.439×10^{-08}	1.702×10^{-10}	2.536×10^{-08}	2.620×10^{-08}	4.732×10^{-10}	$<$
	2	3.274×10^{-04}	3.284×10^{-04}	6.104×10^{-07}	3.288×10^{-04}	3.300×10^{-04}	6.429×10^{-07}	$<$
	5	7.547×10^{-05}	7.591×10^{-05}	1.745×10^{-07}	7.562×10^{-05}	7.604×10^{-05}	2.311×10^{-07}	$<$
	7	9.910×10^{-05}	9.946×10^{-05}	2.452×10^{-07}	9.916×10^{-05}	9.952×10^{-05}	2.227×10^{-07}	$\leq$
	9	3.383×10^{-04}	3.408×10^{-04}	1.274×10^{-06}	3.374×10^{-04}	3.408×10^{-04}	1.440×10^{-06}	$\geq$
QAP	0	152152.00	152442.45	183.71	153496.00	154560.39	555.83	$<$
	6	501575938.00	504059733.39	1076615.21	510503407.00	514628544.29	2557557.14	$<$
	7	44847954.00	44889564.58	19558.50	44898282.00	44949685.10	43215.42	$<$
	8	8150552.00	8172331.68	12316.38	8260506.00	8314086.58	32473.82	$<$
	9	273066.00	273538.45	223.66	274800.00	275459.68	582.77	$<$

Table 1

(b) Results and pairwise comparison of HAMSTA to AILLA (cross-domain tuned) and HAMSTA to TA (cross-domain tuned).

			AILLA				TA		
P.	Inst.	Best	Mean	std.	vs.	Best	Mean	Std.	vs.
BP	1	0.0120	0.0175	4.777×10^{-03}	$<$	0.0399	0.0409	1.254×10^{-03}	$<$
	7	0.0198	0.0304	5.717×10^{-03}	$<$	0.0766	0.0803	2.737×10^{-03}	$<$
	9	0.0201	0.0234	1.255×10^{-03}	$<$	0.0397	0.0443	1.978×10^{-03}	$<$
	10	0.1089	0.1126	2.217×10^{-03}	$<$	0.1261	0.1276	2.733×10^{-04}	$<$
	11	0.0338	0.0374	1.805×10^{-03}	$<$	0.0595	0.0646	2.552×10^{-03}	$<$
FS	1	6238.00	6271.19	17.77	$>$	6374.00	6390.65	6.73	$<$
	3	6323.00	6356.48	17.72	$\leq$	6410.00	6424.65	8.98	$<$
	8	26739.00	26800.03	39.63	$>$	27196.00	27303.35	56.86	$<$
	10	11335.00	11400.23	37.99	$>$	11685.00	11731.81	17.42	$<$
	11	26575.00	26646.26	44.04	$\geq$	26948.00	27076.84	68.58	$<$
PS	5	379.00	928.87	334.87	$<$	688.00	1223.42	364.82	$<$
	8	53408.00	56430.29	1846.62	$<$	58640.00	60041.94	885.82	$<$
	9	85903.00	97238.45	4352.57	$<$	93131.00	99145.97	3507.66	$<$
	10	1659.00	2186.68	503.07	$<$	2165.00	3036.16	713.04	$<$
	11	385.00	1884.45	1186.78	$<$	1805.00	4550.13	1403.64	$<$
SAT	3	3.00	6.77	1.86	$>$	21.00	30.10	5.64	$<$
	4	2.00	6.23	1.84	$>$	17.00	40.48	7.38	$<$
	5	6.00	14.42	7.99	$>$	44.00	54.94	5.88	$<$
	10	2.00	4.87	1.84	$>$	22.00	27.65	3.89	$<$
	11	7.00	7.77	0.76	$>$	9.00	12.84	2.03	$<$
TSP	0	54383.60	57731.16	2554.68	$>$	55135.94	59136.21	2682.78	$>$
	2	7506.15	7765.92	171.21	$>$	7851.89	8077.42	162.08	$<$
	6	59207.47	61460.80	1304.44	$<$	59784.43	61878.83	1299.80	$<$
	7	76440.29	78715.32	837.89	$<$	76940.24	79086.41	829.82	$<$
	8	24802030.05	25064938.08	140462.39	$<$	24807166.51	25074146.64	139687.77	$<$
VRP	1	31513.36	32043.03	479.92	$<$	20866.39	20956.42	268.24	$<$
	2	19351.94	20901.10	576.16	$<$	13569.79	14607.16	276.69	$<$
	5	561560.43	579930.26	8576.51	$<$	330085.32	348607.34	8178.32	$<$
	6	263589.77	269296.82	2783.49	$<$	174754.74	181754.88	3415.29	$<$
	9	380550.51	391457.03	6913.66	$<$	231780.28	240570.19	4942.75	$<$
KP	0	1.016×10^{-05}	1.027×10^{-05}	5.000×10^{-08}	$>$	1.034×10^{-05}	1.034×10^{-05}	5.166×10^{-21}	$\equiv$
	1	2.494×10^{-06}	2.679×10^{-06}	7.107×10^{-08}	$>$	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$
	3	3.917×10^{-06}	4.001×10^{-06}	3.543×10^{-08}	$>$	5.202×10^{-06}	5.202×10^{-06}	8.610×10^{-22}	$<$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.119×10^{-06}	1.126×10^{-06}	3.451×10^{-09}	$>$	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$
MAC	0	3.150×10^{-08}	3.340×10^{-08}	1.077×10^{-09}	$<$	2.403×10^{-08}	2.420×10^{-08}	1.207×10^{-10}	$>$
	2	3.464×10^{-04}	3.483×10^{-04}	1.359×10^{-06}	$<$	3.441×10^{-04}	3.484×10^{-04}	1.983×10^{-06}	$<$
	5	8.033×10^{-05}	8.102×10^{-05}	3.175×10^{-07}	$<$	8.064×10^{-05}	8.094×10^{-05}	2.124×10^{-07}	$<$
	7	1.087×10^{-04}	1.093×10^{-04}	3.961×10^{-07}	$<$	1.085×10^{-04}	1.094×10^{-04}	3.074×10^{-07}	$<$
	9	4.562×10^{-04}	4.738×10^{-04}	6.511×10^{-06}	$<$	4.566×10^{-04}	4.725×10^{-04}	7.390×10^{-06}	$<$
QAP	0	152178.00	152861.16	564.60	$<$	160296.00	160816.77	223.02	$<$
	6	505536415.00	511318577.84	3434527.65	$<$	530734376.00	533404490.61	1211399.48	$<$
	7	44812768.00	44863273.16	27230.70	$>$	45112580.00	45205163.48	45724.22	$<$
	8	8153134.00	8207670.32	32794.51	$<$	8875552.00	8936440.90	20031.93	$<$
	9	273336.00	273895.03	401.01	$<$	285808.00	286387.35	262.84	$<$

Table 1

(c) Results and pairwise comparison of HAMSTA to GD (cross-domain tuned) and HAMSTA to AILTA (cross-domain tuned).

		GD				AILTA			
P.	Inst.	Best	Mean	std.	vs.	Best	Mean	Std.	vs.
	1	3.554×10^{-03}	0.0695	0.0409	$<$	0.0170	0.0178	2.152×10^{-04}	$<$
	7	8.434×10^{-03}	0.1373	0.0818	$<$	0.0194	0.0215	1.425×10^{-03}	$<$
BP	9	0.0199	0.0469	0.0164	$<$	0.0210	0.0219	7.019×10^{-04}	$<$
	10	0.1086	0.1235	9.136×10^{-03}	$<$	0.1104	0.1128	1.076×10^{-03}	$<$
	11	0.0315	0.0782	0.0282	$<$	0.0337	0.0362	1.242×10^{-03}	$<$
	1	6268.00	6486.94	137.41	$<$	6342.00	6365.26	10.45	$<$
	3	6316.00	6523.29	120.23	$<$	6387.00	6406.00	7.19	$<$
FS	8	26752.00	27164.52	224.76	$<$	27188.00	27273.94	34.88	$<$
	10	11359.00	11663.26	167.62	$<$	11555.00	11595.32	16.39	$<$
	11	26619.00	26959.81	204.57	$<$	26948.00	27045.90	42.35	$<$
	5	46.00	442.19	647.91	$<$	688.00	1207.00	363.83	$<$
	8	44875.00	52466.45	5148.52	$<$	58640.00	60041.94	885.82	$<$
PS	9	41155.00	69440.35	20640.20	$>$	93131.00	99145.97	3507.66	$<$
	10	1555.00	2043.06	718.90	$<$	2165.00	3036.16	713.04	$<$
	11	406.00	1728.84	2072.18	$<$	1805.00	4550.13	1403.64	$<$
	3	20.00	178.35	96.60	$<$	21.00	30.10	5.64	$<$
	4	19.00	190.13	98.60	$<$	17.00	40.42	7.46	$<$
SAT	5	45.00	265.84	137.81	$<$	44.00	54.94	5.88	$<$
	10	19.00	236.90	136.05	$<$	22.00	27.65	3.89	$<$
	11	11.00	80.16	43.28	$<$	9.00	12.81	2.04	$<$
	0	58245.37	62601.91	2617.32	$<$	54217.61	57590.58	2033.13	$>$
	2	7853.79	8208.88	169.00	$<$	8042.55	8274.93	130.09	$<$
TSP	6	58320.84	61319.59	1609.35	$<$	59784.43	61871.75	1295.11	$<$
	7	75845.86	78406.90	1346.79	$<$	76940.24	79086.82	829.38	$<$
	8	24790295.78	25068524.73	141123.48	$<$	24807166.51	25074153.26	139700.45	$<$
	1	22808.65	37146.26	8984.29	$<$	20682.28	20706.17	11.76	$\geq$
	2	14522.18	26915.35	7327.28	$<$	12342.36	13248.17	291.27	$\geq$
VRP	5	195424.25	470794.60	172724.15	$<$	250893.36	258435.16	3578.63	$<$
	6	108747.20	266081.34	100237.43	$<$	69874.24	74217.61	1683.97	$>$
	9	162830.24	325426.90	104611.01	$<$	162347.78	166383.05	1730.67	$<$
	0	1.027×10^{-05}	1.033×10^{-05}	1.836×10^{-08}	$>$	1.034×10^{-05}	1.034×10^{-05}	5.166×10^{-21}	$\equiv$
	1	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$
KP	3	4.969×10^{-06}	5.152×10^{-06}	7.870×10^{-08}	$<$	5.202×10^{-06}	5.202×10^{-06}	8.610×10^{-22}	$<$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$
	0	2.537×10^{-08}	3.096×10^{-08}	3.085×10^{-09}	$<$	2.417×10^{-08}	2.433×10^{-08}	1.085×10^{-10}	$\geq$
	2	3.291×10^{-04}	3.433×10^{-04}	8.754×10^{-06}	$<$	3.298×10^{-04}	3.308×10^{-04}	4.405×10^{-07}	$<$
MAC	5	7.569×10^{-05}	7.942×10^{-05}	2.305×10^{-06}	$<$	7.616×10^{-05}	7.624×10^{-05}	6.780×10^{-08}	$<$
	7	9.930×10^{-05}	1.065×10^{-04}	4.514×10^{-06}	$<$	1.005×10^{-04}	1.008×10^{-04}	1.277×10^{-07}	$<$
	9	3.399×10^{-04}	4.356×10^{-04}	6.117×10^{-05}	$<$	3.571×10^{-04}	3.598×10^{-04}	1.365×10^{-06}	$<$
	0	154048.00	166397.87	7648.67	$<$	153258.00	153472.71	95.99	$<$
	6	510736883.00	585264322.32	46495690.44	$<$	501964419.00	503384010.29	762676.56	$>$
QAP	7	44926378.00	47850726.45	1880866.58	$<$	44982122.00	45030656.65	22561.41	$<$
	8	8258620.00	9113265.55	515023.69	$<$	8245986.00	8259894.32	7525.70	$<$
	9	274982.00	287382.58	7887.73	$<$	274574.00	274763.87	94.78	$<$

Table 1

(d) Results and pairwise comparison of HAMSTA to NA (cross-domain tuned) and HAMSTA to SA (cross-domain tuned).

			NA				SA		
P.	Inst.	Best	Mean	std.	vs.	Best	Mean	Std.	vs.
	1	0.0537	0.0577	2.308×10^{-03}	$<$	0.0539	0.0602	3.133×10^{-03}	$<$
	7	0.1787	0.1820	1.412×10^{-03}	$<$	0.0890	0.0998	4.907×10^{-03}	$<$
BP	9	0.0342	0.0368	8.457×10^{-04}	$<$	0.0319	0.0356	1.631×10^{-03}	$<$
	10	0.1270	0.1272	7.674×10^{-05}	$<$	0.1257	0.1272	3.088×10^{-04}	$<$
	11	0.0633	0.0656	1.185×10^{-03}	$<$	0.0602	0.0646	2.270×10^{-03}	$<$
	1	6488.00	6576.29	43.62	$<$	6330.00	6360.00	17.41	$<$
	3	6525.00	6611.55	41.42	$<$	6389.00	6409.94	12.34	$<$
FS	8	27196.00	27303.77	56.89	$<$	26943.00	27014.42	43.57	$<$
	10	11685.00	11768.61	47.88	$<$	11509.00	11541.71	19.51	$<$
	11	26948.00	27076.84	68.58	$<$	26718.00	26822.68	43.34	$<$
	5	42.00	47.77	3.61	$\geq$	36.00	52.94	5.59	$<$
	8	45775.00	48672.52	1638.52	$\leq$	43421.00	49034.87	2356.03	$\leq$
PS	9	43290.00	51132.55	4611.09	$>$	53979.00	70554.10	7434.28	$>$
	10	1495.00	1612.23	62.27	$>$	1530.00	1684.81	89.19	$\leq$
	11	410.00	455.26	30.17	$>$	430.00	562.39	222.60	$\leq$
	3	165.00	171.74	3.49	$<$	2.00	6.87	2.62	$>$
	4	170.00	180.61	4.55	$<$	2.00	4.42	1.48	$>$
SAT	5	246.00	255.06	4.07	$<$	4.00	7.97	3.18	$>$
	10	228.00	237.03	3.73	$<$	5.00	8.61	2.14	$>$
	11	79.00	82.90	1.70	$<$	7.00	8.55	1.06	$>$
	0	59561.29	63039.82	2480.25	$<$	56352.90	59627.33	2125.51	$>$
	2	8042.55	8279.75	131.14	$<$	8042.55	8279.75	131.14	$<$
TSP	6	59784.43	61888.08	1299.22	$<$	59784.43	61888.08	1299.22	$<$
	7	76940.24	79086.89	829.40	$<$	76940.24	79086.89	829.40	$<$
	8	24807166.51	25074153.26	139700.45	$<$	24807166.51	25074153.26	139700.45	$<$
	1	37828.93	42973.44	1831.25	$<$	20821.33	20959.15	316.39	$<$
	2	28730.33	31213.79	973.30	$<$	14626.95	14696.52	202.41	$<$
VRP	5	561560.43	580589.07	8618.93	$<$	242985.46	259613.09	9390.93	$<$
	6	310641.97	328147.37	6404.38	$<$	72934.36	81755.23	3345.95	$>$
	9	380645.18	391475.87	6840.35	$<$	177305.28	184750.88	3726.94	$<$
	0	1.015×10^{-05}	1.028×10^{-05}	4.283×10^{-08}	$>$	1.026×10^{-05}	1.032×10^{-05}	2.729×10^{-08}	$>$
	1	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$	2.585×10^{-06}	2.807×10^{-06}	1.035×10^{-07}	$>$
KP	3	4.501×10^{-06}	4.593×10^{-06}	4.197×10^{-08}	$<$	4.094×10^{-06}	4.240×10^{-06}	4.379×10^{-08}	$>$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$	1.128×10^{-06}	1.149×10^{-06}	6.763×10^{-09}	$>$
	0	3.174×10^{-08}	3.387×10^{-08}	1.204×10^{-09}	$<$	2.405×10^{-08}	2.425×10^{-08}	1.109×10^{-10}	$>$
	2	3.463×10^{-04}	3.490×10^{-04}	1.745×10^{-06}	$<$	3.266×10^{-04}	3.271×10^{-04}	3.068×10^{-07}	$>$
MAC	5	8.030×10^{-05}	8.086×10^{-05}	2.627×10^{-07}	$<$	7.488×10^{-05}	7.498×10^{-05}	9.174×10^{-08}	$>$
	7	1.083×10^{-04}	1.092×10^{-04}	3.433×10^{-07}	$<$	9.746×10^{-05}	9.767×10^{-05}	1.134×10^{-07}	$>$
	9	4.587×10^{-04}	4.709×10^{-04}	6.018×10^{-06}	$<$	3.358×10^{-04}	3.381×10^{-04}	1.097×10^{-06}	$>$
	0	167756.00	168563.55	287.79	$<$	152026.00	152144.13	88.02	$>$
	6	594728166.00	598287549.65	1383469.74	$<$	501112629.00	502515063.61	751228.99	$>$
QAP	7	47701294.00	47927420.00	107547.32	$<$	44843748.00	44886308.52	20954.23	$\geq$
	8	9242016.00	9280370.39	14140.28	$<$	8142722.00	8151401.10	4453.25	$>$
	9	288942.00	289552.26	238.53	$<$	273048.00	273145.87	93.19	$>$

Table 1

(e) Results and pairwise comparison of HAMSTA to SARH (cross-domain tuned).

		SARH			
P.	Inst.	Best	Mean	std.	vs.
BP	1	0.0586	0.0654	4.162×10^{-03}	$<$
	7	0.1137	0.1244	5.582×10^{-03}	$<$
	9	0.0333	0.0366	1.882×10^{-03}	$<$
	10	0.1276	0.1277	7.678×10^{-05}	$<$
	11	0.0632	0.0672	2.294×10^{-03}	$<$
FS	1	6488.00	6575.55	41.96	$<$
	3	6525.00	6609.06	34.43	$<$
	8	27196.00	27303.77	56.89	$<$
	10	11685.00	11768.61	47.88	$<$
	11	26948.00	27076.84	68.58	$<$
PS	5	39.00	51.94	5.16	$\leq$
	8	45419.00	49514.90	2398.62	$<$
	9	46985.00	65881.16	8632.22	$>$
	10	1540.00	1657.55	70.98	$\leq$
	11	375.00	491.45	47.50	$>$
SAT	3	7.00	12.00	2.59	$>$
	4	5.00	8.10	1.40	$>$
	5	9.00	15.29	6.81	$>$
	10	6.00	10.94	2.00	$\geq$
	11	7.00	9.26	1.24	$>$
TSP	0	59561.29	62918.06	2358.57	$<$
	2	8042.55	8279.75	131.14	$<$
	6	59784.43	61888.08	1299.22	$<$
	7	76940.24	79086.89	829.40	$<$
	8	24807166.51	25074153.26	139700.45	$<$
VRP	1	22043.00	23988.25	638.97	$<$
	2	12272.87	12469.71	387.51	$>$
	5	362890.27	380664.68	8103.15	$<$
	6	129738.02	136064.05	4072.01	$<$
	9	239703.99	254366.27	6435.73	$<$
KP	0	1.016×10^{-05}	1.030×10^{-05}	3.754×10^{-08}	$>$
	1	2.449×10^{-06}	2.811×10^{-06}	1.095×10^{-07}	$>$
	3	4.112×10^{-06}	4.232×10^{-06}	3.378×10^{-08}	$>$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.138×10^{-06}	1.148×10^{-06}	5.050×10^{-09}	$>$
MAC	0	2.519×10^{-08}	2.550×10^{-08}	1.328×10^{-10}	$<$
	2	3.351×10^{-04}	3.364×10^{-04}	6.044×10^{-07}	$<$
	5	7.672×10^{-05}	7.706×10^{-05}	1.428×10^{-07}	$<$
	7	1.024×10^{-04}	1.029×10^{-04}	2.076×10^{-07}	$<$
	9	4.102×10^{-04}	4.152×10^{-04}	1.900×10^{-06}	$<$
QAP	0	153744.00	154040.97	144.36	$<$
	6	506229031.00	508054332.48	1064190.32	$<$
	7	45224240.00	45344817.74	54257.52	$<$
	8	8282630.00	8299738.26	9292.22	$<$
	9	274924.00	275122.90	148.15	$<$

Table 2: Results and pairwise comparisons using Wilcoxon Signed Rank test (CI = 95%) for each problem instance of HAMSTA (cross-domain tuned) to the per-domain tuned move acceptance methods where $<$ ($>$) denotes that HAMSTA performs better (worse) than the associated move acceptance method, $\leq$ ($\geq$) denotes that HAMSTA does not perform statistically significantly different from the associated method, but performs on (mean) average better (worse). $\equiv$ denotes that HAMSTA and the associated algorithm have equivalent performance.

(a) Results of HAMSTA and pairwise comparison to IE.

		HAMSTA			IE			
P.	Inst.	Best	Mean	std.	Best	Mean	Std.	vs.
BP	1	3.394×10^{-03}	5.258×10^{-03}	1.719×10^{-03}	3.392×10^{-03}	4.450×10^{-03}	1.526×10^{-03}	$>$
	7	5.781×10^{-03}	8.543×10^{-03}	9.380×10^{-04}	5.720×10^{-03}	7.994×10^{-03}	7.724×10^{-04}	$>$
	9	0.0144	0.0157	7.918×10^{-04}	0.0123	0.0137	6.505×10^{-04}	$>$
	10	0.1086	0.1091	6.849×10^{-04}	0.1085	0.1088	5.215×10^{-04}	$>$
	11	0.0263	0.0276	9.894×10^{-04}	0.0225	0.0244	9.532×10^{-04}	$>$
FS	1	6250.00	6280.06	17.15	6236.00	6279.00	23.81	$\geq$
	3	6310.00	6352.65	21.37	6319.00	6345.90	19.50	$\geq$
	8	26754.00	26820.19	33.91	26750.00	26804.32	33.62	$>$
	10	11352.00	11414.68	37.48	11320.00	11385.68	32.93	$>$
	11	26591.00	26649.32	43.30	26555.00	26638.87	33.82	$\geq$
PS	5	38.00	49.68	6.00	688.00	1223.42	364.82	$<$
	8	43562.00	48469.23	2050.65	58640.00	60041.94	885.82	$<$
	9	63518.00	83842.81	6045.60	93131.00	99145.97	3507.66	$<$
	10	1520.00	1653.94	65.48	2165.00	3036.16	713.04	$<$
	11	415.00	513.74	55.94	1805.00	4550.13	1403.64	$<$
SAT	3	7.00	13.61	3.87	21.00	30.10	5.64	$<$
	4	4.00	13.94	6.10	17.00	40.42	7.46	$<$
	5	7.00	23.81	12.54	44.00	54.94	5.88	$<$
	10	6.00	11.06	2.98	22.00	27.65	3.89	$<$
	11	7.00	10.13	1.36	9.00	12.81	2.04	$<$
TSP	0	56845.65	60849.40	2773.94	58241.52	61397.45	2654.51	$<$
	2	7806.49	8013.44	136.49	7806.49	8015.58	137.54	$\leq$
	6	57843.55	59683.69	1073.44	57783.36	59660.40	1080.48	$>$
	7	74894.97	76674.23	697.67	74837.82	76636.78	703.42	$>$
	8	24787330.82	25046171.88	142593.00	24766167.28	25027169.31	142422.07	$>$
VRP	1	20672.57	20868.33	382.43	22763.06	24089.92	877.80	$<$
	2	12283.32	13382.69	505.43	14552.51	16155.79	652.04	$<$
	5	195424.25	205064.48	4191.12	195424.25	205237.19	4229.97	$\leq$
	6	107792.49	112929.04	3195.38	107791.35	113050.05	3229.83	$<$
	9	161371.90	165137.43	2594.57	161714.36	165420.08	2437.13	$<$
KP	0	1.034×10^{-05}	1.034×10^{-05}	5.166×10^{-21}	1.034×10^{-05}	1.034×10^{-05}	5.166×10^{-21}	$\equiv$
	1	2.673×10^{-06}	3.206×10^{-06}	1.648×10^{-07}	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$
	3	4.163×10^{-06}	4.315×10^{-06}	7.118×10^{-08}	5.202×10^{-06}	5.202×10^{-06}	8.610×10^{-22}	$<$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.139×10^{-06}	1.153×10^{-06}	6.587×10^{-09}	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$
MAC	0	2.416×10^{-08}	2.439×10^{-08}	1.702×10^{-10}	2.536×10^{-08}	2.620×10^{-08}	4.732×10^{-10}	$<$
	2	3.274×10^{-04}	3.284×10^{-04}	6.104×10^{-07}	3.288×10^{-04}	3.300×10^{-04}	6.429×10^{-07}	$<$
	5	7.547×10^{-05}	7.591×10^{-05}	1.745×10^{-07}	7.562×10^{-05}	7.604×10^{-05}	2.311×10^{-07}	$<$
	7	9.910×10^{-05}	9.946×10^{-05}	2.452×10^{-07}	9.916×10^{-05}	9.952×10^{-05}	2.227×10^{-07}	$\leq$
	9	3.383×10^{-04}	3.408×10^{-04}	1.274×10^{-06}	3.374×10^{-04}	3.408×10^{-04}	1.440×10^{-06}	$\geq$
QAP	0	152152.00	152442.45	183.71	153496.00	154560.39	555.83	$<$
	6	501575938.00	504059733.39	1076615.21	510503407.00	514628544.29	2557557.14	$<$
	7	44847954.00	44889564.58	19558.50	44898282.00	44949685.10	43215.42	$<$
	8	8150552.00	8172331.68	12316.38	8260506.00	8314086.58	32473.82	$<$
	9	273066.00	273538.45	223.66	274800.00	275459.68	582.77	$<$

Table 2

(b) Results and pairwise comparison of HAMSTA to TA (per-domain tuned) and HAMSTA to GD (per-domain tuned).

		TA				GD			
P.	Inst.	Best	Mean	std.	vs.	Best	Mean	Std.	vs.
	1	3.483×10^{-03}	6.307×10^{-03}	1.557×10^{-03}	$<$	3.532×10^{-03}	0.0693	0.0417	$<$
	7	8.364×10^{-03}	0.0106	1.929×10^{-03}	$<$	8.489×10^{-03}	0.1373	0.0818	$<$
BP	9	0.0155	0.0190	2.995×10^{-03}	$<$	0.0147	0.0460	0.0179	$<$
	10	0.1086	0.1095	7.022×10^{-04}	$<$	0.1086	0.1235	9.224×10^{-03}	$<$
	11	0.0263	0.0348	5.220×10^{-03}	$<$	0.0273	0.0778	0.0290	$<$
	1	6236.00	6288.23	28.73	$<$	6268.00	6494.16	133.66	$<$
	3	6319.00	6350.29	19.32	$\geq$	6319.00	6524.42	119.67	$<$
FS	8	26748.00	26811.74	27.92	$\geq$	26751.00	27164.45	225.25	$<$
	10	11354.00	11418.19	32.73	$\leq$	11359.00	11663.39	166.80	$<$
	11	26564.00	26651.48	36.94	$\leq$	26609.00	26959.90	204.60	$<$
	5	36.00	47.74	5.34	$\geq$	46.00	442.45	647.75	$<$
	8	44875.00	49445.65	2023.05	$<$	44875.00	52466.45	5148.52	$<$
PS	9	36483.00	53895.10	7559.25	$>$	38413.00	66373.32	22189.62	$>$
	10	1435.00	1904.84	354.74	$<$	1555.00	2046.06	717.18	$<$
	11	406.00	513.16	61.47	$\geq$	406.00	1723.97	2075.02	$<$
	3	21.00	30.10	5.64	$<$	22.00	177.42	96.41	$<$
	4	17.00	40.42	7.46	$<$	15.00	188.68	99.43	$<$
SAT	5	44.00	54.94	5.88	$<$	50.00	266.55	136.67	$<$
	10	22.00	27.65	3.89	$<$	21.00	236.55	135.24	$<$
	11	9.00	12.81	2.04	$<$	11.00	80.48	42.91	$<$
	0	55135.94	59130.90	2690.62	$>$	58245.37	62600.29	2620.74	$<$
	2	7836.62	8099.48	152.40	$<$	7853.79	8208.75	168.79	$<$
TSP	6	59784.43	61878.83	1299.80	$<$	58320.84	61319.59	1609.35	$<$
	7	76940.24	79086.41	829.82	$<$	75736.09	78400.29	1358.42	$<$
	8	24807166.51	25074146.64	139687.77	$<$	24787330.82	25067737.25	141473.46	$<$
	1	20662.90	20821.67	340.44	$\geq$	22801.07	37096.93	8800.84	$<$
	2	12279.21	12731.89	505.48	$>$	15542.93	27049.66	7048.68	$<$
VRP	5	218239.66	231251.98	5907.40	$<$	195425.01	470794.62	172724.12	$<$
	6	71088.61	75107.91	2195.05	$>$	108747.20	266082.40	100235.70	$<$
	9	166317.05	170089.49	2119.39	$<$	162819.53	325424.46	104614.93	$<$
	0	1.026×10^{-05}	1.033×10^{-05}	1.917×10^{-08}	$>$	1.024×10^{-05}	1.033×10^{-05}	2.367×10^{-08}	$>$
	1	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$
KP	3	4.969×10^{-06}	5.142×10^{-06}	8.699×10^{-08}	$<$	4.897×10^{-06}	5.160×10^{-06}	8.750×10^{-08}	$<$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$
	0	2.431×10^{-08}	2.492×10^{-08}	3.463×10^{-10}	$<$	2.574×10^{-08}	3.084×10^{-08}	3.096×10^{-09}	$<$
	2	3.358×10^{-04}	3.368×10^{-04}	4.526×10^{-07}	$<$	3.292×10^{-04}	3.429×10^{-04}	8.402×10^{-06}	$<$
MAC	5	8.065×10^{-05}	8.105×10^{-05}	2.370×10^{-07}	$<$	7.579×10^{-05}	7.954×10^{-05}	2.297×10^{-06}	$<$
	7	1.089×10^{-04}	1.094×10^{-04}	3.400×10^{-07}	$<$	9.918×10^{-05}	1.065×10^{-04}	4.620×10^{-06}	$<$
	9	3.401×10^{-04}	3.442×10^{-04}	1.783×10^{-06}	$<$	3.383×10^{-04}	4.357×10^{-04}	6.178×10^{-05}	$<$
	0	152088.00	152249.10	108.56	$>$	154522.00	166450.77	7410.60	$<$
	6	503773903.00	506671553.06	1496768.06	$<$	506662705.00	585301816.26	46006089.58	$<$
QAP	7	44849624.00	44894702.52	22110.41	$\leq$	44893236.00	47878937.16	1913438.65	$<$
	8	8201886.00	8215772.52	6785.94	$<$	8252552.00	9117550.06	526932.34	$<$
	9	274006.00	274177.29	75.83	$<$	275188.00	287469.48	7556.39	$<$

Table 2

(c) Results and pairwise comparison of HAMSTA to NA and HAMSTA to SA (per-domain tuned).

			NA				SA		
P.	Inst.	Best	Mean	std.	vs.	Best	Mean	Std.	vs.
BP	1	0.0537	0.0577	2.308×10^{-03}	$<$	3.479×10^{-03}	6.200×10^{-03}	1.627×10^{-03}	$<$
	7	0.1787	0.1820	1.412×10^{-03}	$<$	8.068×10^{-03}	9.943×10^{-03}	1.398×10^{-03}	$<$
	9	0.0342	0.0368	8.457×10^{-04}	$<$	0.0147	0.0185	2.004×10^{-03}	$<$
	10	0.1270	0.1272	7.674×10^{-05}	$<$	0.1085	0.1097	1.668×10^{-03}	$<$
	11	0.0633	0.0656	1.185×10^{-03}	$<$	0.0247	0.0324	3.474×10^{-03}	$<$
FS	1	6488.00	6576.29	43.62	$<$	6247.00	6301.16	21.98	$<$
	3	6525.00	6611.55	41.42	$<$	6310.00	6348.48	21.00	$\geq$
	8	27196.00	27303.77	56.89	$<$	26745.00	26821.45	42.07	$\leq$
	10	11685.00	11768.61	47.88	$<$	11323.00	11405.71	38.68	$\geq$
	11	26948.00	27076.84	68.58	$<$	26587.00	26664.39	41.61	$<$
PS	5	42.00	47.77	3.61	$\geq$	44.00	50.87	4.19	$\leq$
	8	45775.00	48672.52	1638.52	$\leq$	43421.00	49406.19	2365.76	$<$
	9	43290.00	51132.55	4611.09	$>$	41412.00	52587.00	7238.96	$>$
	10	1495.00	1612.23	62.27	$>$	1490.00	1658.97	95.04	$\leq$
	11	410.00	455.26	30.17	$>$	430.00	497.81	40.06	$\geq$
SAT	3	165.00	171.74	3.49	$<$	4.00	7.52	1.95	$>$
	4	170.00	180.61	4.55	$<$	2.00	5.32	1.76	$>$
	5	246.00	255.06	4.07	$<$	6.00	8.81	1.82	$>$
	10	228.00	237.03	3.73	$<$	4.00	8.84	2.75	$>$
	11	79.00	82.90	1.70	$<$	7.00	8.39	0.99	$>$
TSP	0	59561.29	63039.82	2480.25	$<$	59561.29	62968.50	2467.50	$<$
	2	8042.55	8279.75	131.14	$<$	8042.55	8271.40	136.30	$<$
	6	59784.43	61888.08	1299.22	$<$	57783.36	60044.79	1293.20	$<$
	7	76940.24	79086.89	829.40	$<$	74909.60	76915.97	863.09	$<$
	8	24807166.51	25074153.26	139700.45	$<$	24790896.18	25061207.86	143130.85	$<$
VRP	1	37828.93	42973.44	1831.25	$<$	21954.89	22821.99	456.43	$<$
	2	28730.33	31213.79	973.30	$<$	12266.70	12520.83	438.75	$>$
	5	561560.43	580589.07	8618.93	$<$	320333.52	340477.58	7654.09	$<$
	6	310641.97	328147.37	6404.38	$<$	114355.45	120760.65	3181.84	$<$
	9	380645.18	391475.87	6840.35	$<$	222240.82	228243.70	4196.11	$<$
KP	0	1.015×10^{-05}	1.028×10^{-05}	4.283×10^{-08}	$>$	1.023×10^{-05}	1.030×10^{-05}	3.153×10^{-08}	$>$
	1	3.349×10^{-06}	3.349×10^{-06}	1.722×10^{-21}	$<$	2.608×10^{-06}	2.797×10^{-06}	8.210×10^{-08}	$>$
	3	4.501×10^{-06}	4.593×10^{-06}	4.197×10^{-08}	$<$	4.146×10^{-06}	4.216×10^{-06}	2.520×10^{-08}	$>$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.298×10^{-06}	1.298×10^{-06}	6.458×10^{-22}	$<$	1.138×10^{-06}	1.146×10^{-06}	4.263×10^{-09}	$>$
MAC	0	3.174×10^{-08}	3.387×10^{-08}	1.204×10^{-09}	$<$	2.559×10^{-08}	2.620×10^{-08}	3.929×10^{-10}	$<$
	2	3.463×10^{-04}	3.490×10^{-04}	1.745×10^{-06}	$<$	3.277×10^{-04}	3.303×10^{-04}	8.836×10^{-07}	$<$
	5	8.030×10^{-05}	8.086×10^{-05}	2.627×10^{-07}	$<$	7.563×10^{-05}	7.604×10^{-05}	1.819×10^{-07}	$<$
	7	1.083×10^{-04}	1.092×10^{-04}	3.433×10^{-07}	$<$	9.905×10^{-05}	9.943×10^{-05}	2.024×10^{-07}	$\geq$
	9	4.587×10^{-04}	4.709×10^{-04}	6.018×10^{-06}	$<$	3.390×10^{-04}	3.408×10^{-04}	1.128×10^{-06}	$\leq$
QAP	0	167756.00	168563.55	287.79	$<$	153192.00	153509.23	128.69	$<$
	6	594728166.00	598287549.65	1383469.74	$<$	504241335.00	506315153.74	1043454.22	$<$
	7	47701294.00	47927420.00	107547.32	$<$	45091456.00	45203476.90	43572.62	$<$
	8	9242016.00	9280370.39	14140.28	$<$	8244208.00	8262139.42	6028.39	$<$
	9	288942.00	289552.26	238.53	$<$	274392.00	274563.74	100.85	$<$

Table 2

(d) Results and pairwise comparison of HAMSTA to SARH (per-domain tuned).

		SARH			
P.	Inst.	Best	Mean	std.	vs.
BP	1	0.0499	0.0633	3.935×10^{-03}	$<$
	7	0.1040	0.1125	5.236×10^{-03}	$<$
	9	0.0354	0.0378	1.596×10^{-03}	$<$
	10	0.1274	0.1276	8.616×10^{-05}	$<$
	11	0.0637	0.0673	2.277×10^{-03}	$<$
FS	1	6260.00	6290.19	21.18	$<$
	3	6319.00	6355.52	15.71	$\leq$
	8	26751.00	26822.81	37.09	$\leq$
	10	11359.00	11411.71	34.13	$\geq$
	11	26580.00	26666.74	41.61	$<$
PS	5	43.00	50.97	4.79	$\leq$
	8	45495.00	49897.23	2215.78	$<$
	9	39787.00	53399.94	7501.88	$>$
	10	1509.00	1639.97	74.10	$\geq$
	11	416.00	493.16	37.41	$>$
SAT	3	2.00	5.19	1.85	$>$
	4	2.00	3.58	1.12	$>$
	5	3.00	6.13	1.57	$>$
	10	2.00	6.45	2.31	$>$
	11	7.00	7.94	0.89	$>$
TSP	0	59561.29	63039.82	2480.25	$<$
	2	8032.09	8257.63	120.76	$<$
	6	58320.84	60288.30	1513.36	$<$
	7	74837.82	76672.56	707.29	$\geq$
	8	24787319.65	25044391.34	141843.72	$\geq$
VRP	1	21954.89	22821.99	456.43	$<$
	2	12266.70	12520.83	438.75	$>$
	5	320333.52	340477.58	7654.09	$<$
	6	114355.45	120760.65	3181.84	$<$
	9	222240.82	228243.70	4196.11	$<$
KP	0	1.014×10^{-05}	1.029×10^{-05}	5.173×10^{-08}	$>$
	1	2.668×10^{-06}	2.807×10^{-06}	6.519×10^{-08}	$>$
	3	4.095×10^{-06}	4.225×10^{-06}	3.568×10^{-08}	$>$
	5	8.160×10^{-07}	8.160×10^{-07}	4.305×10^{-22}	$\equiv$
	8	1.133×10^{-06}	1.145×10^{-06}	4.433×10^{-09}	$>$
MAC	0	2.460×10^{-08}	2.483×10^{-08}	1.080×10^{-10}	$<$
	2	3.272×10^{-04}	3.282×10^{-04}	3.867×10^{-07}	$\geq$
	5	7.505×10^{-05}	7.517×10^{-05}	8.421×10^{-08}	$>$
	7	9.816×10^{-05}	9.836×10^{-05}	1.143×10^{-07}	$>$
	9	3.529×10^{-04}	3.553×10^{-04}	1.275×10^{-06}	$<$
QAP	0	153238.00	153486.19	118.43	$<$
	6	504846786.00	506413442.23	824052.44	$<$
	7	45049522.00	45191973.23	45339.33	$<$
	8	8250384.00	8261080.06	6368.08	$<$
	9	274242.00	274525.87	112.64	$<$