Universiteit Utrecht

Faculty of Science

Optimizing Optimization

Hyper-Heuristic Approaches for Generating Perturbative Operators through Operator Chaining

Master Thesis

Catalina Blom (5923921)
Computing Science

Supervisor:
Dr. J.M.M. (Johan) van Rooij
Utrecht University

Supervisor:
Dr. J.A. (Han) Hoogeveen
Utrecht University

Supervisor:
Dr. Pepijn Wissing
Consultants in Quantitative Methods

Supervisor:
Arjen Sijtsma MSc
Consultants in Quantitative Methods

July 24, 2024

Acknowledgements

This thesis would not have been possible without the support and guidance of many wonderful people. First and foremost, I would like to express my deepest gratitude to my supervisors: Arjen, for always coming up with new and exciting ideas; Pepijn, for your critical yet fair viewpoint and for keeping me grounded in reality; and Johan, for your consistent support and for looking out for my mental well-being as well as the quality of my thesis.

In addition to this, there are several old and new housemates to whom I owe a lot. First, to my parents, for proofreading my work. Second, to Joanna, for her constant mental support and always being just a phone call or short train ride away. Third, to HJ, for always being willing to help, no matter your state of mind. Finally, to my current housemates, for putting up with me (even when I am sometimes tired, cranky, or stressed) and making our house a home.

I am also very grateful for Veerle, Kasper and Olav. Meeting you made my time in the Computing Science masters not just manageable but also incredibly enjoyable. Thanks to you, I looked forward to attending class every single day. I owe an additional thank you to Veerle, for being crazy enough to sign up for the Nacht van Eindhoven with me, thus introducing me to CQM.

Finally, I owe a special thanks to Aniek and Doris, with whom I shared the final and most stressful phases of the thesis writing process. I now look back on this time as a golden era as opposed to the dark ages it otherwise would have been. I really do not know how I would have managed without you and I am deeply grateful that we were able to go through this experience together.

Abstract

This thesis investigates the development of hyper-heuristics designed to generate perturbative operators through operator chaining, with the objective of enhancing the performance of search heuristics applied to combinatorial optimization problems. Three innovative hyper-heuristic approaches are introduced: Priority Tree Search (PTS), Mining Hyper-Heuristic (M-HH), and Population Mining Hyper-Heuristic (PM-HH). Each approach focuses on creating composite operators, or chains, by combining simpler basis operators to increase the efficiency of the search process.

The effectiveness of these hyper-heuristics is evaluated using the Simulated Annealing (SA) algorithm applied to the Trapdoor Problem. The study demonstrates that the chain structure, even with a limited set of building blocks, can generate operators that significantly enhance the search process compared to the base case. The M-HH and PM-HH approaches are particularly effective in prioritizing and discovering high-quality chains, nearly identifying all chains that improve search quality. While PTS also discovers all potential chains, it is less effective at prioritizing the higher-quality ones.

This research makes a significant contribution to the field of optimization by providing automated methods for generating effective composite operators. These methods reduce the need for extensive manual tuning and domain-specific expertise, showcasing the potential for broad applicability across various combinatorial optimization problems. Furthermore, this study demonstrates the potential of mining-based techniques in identifying effective operators in a limited time.

Contents

Chapter 1

Introduction

Consider the following analogy. You are an experienced traveller, knowledgeable in computer science and mathematics, navigating the seven seas. One day, pirates capture you and take you to an unknown island. They offer you a chance to win your freedom: each day, at sunrise, they place you at a random location on a random island. If you reach the highest point of the island before sunset, you and your crew are free. The catch? You must do this blindfolded, unable to tell if you're stepping up or down. To help guide your moves you have a parrot that tells you your height every 10 minutes.

Recognizing something that resembles a Combinatorial Optimization problem, you decide to employ a Heuristic algorithm. Every 10 minutes, you take a random step and wait for the parrot to report your height. If your height increases or decreases by at most 20 centimeters, you continue from the new location. Otherwise, you step back and try a different direction. If you step into the water, you also restart from a new random location. You repeat this process daily.

Days pass without success. You realize that taking two steps in the same direction often leads to larger height increases. Incorporating this insight, you alternate between single steps and big jumps. The next day, you finally reach the highest point to the applause of the pirates. Removing your blindfold, you see you're on an island connected to others by short bridges, which take a few steps to cross. Freeing your crew, you sail away, reflecting on the importance of proper operator choice in heuristics.

Figure 1.1: A map of the island(s)

Just like the protagonist of this story, this thesis develops several algorithms for the generation of operators for search heuristics. Each of these algorithms relies on the generation of new operators through the composition of existing operators.

This chapter is structured as follows: Section 1.1 provides an overview of the problem, including definitions of Combinatorial Optimization problems, Heuristics, and Hyper-Heuristics. This section sets the stage for understanding the context and significance of the research. Section 1.2 reviews existing literature related to the development and application of operators within heuristics. This includes a discussion of different approaches to operator generation and highlighting gaps that this thesis aims to address. Section 1.3 explores the practical importance of the research, both in terms of specific applications and broader implications for the field of optimization. Section 1.4 details the various steps taken during the research, including the initial attempts and the reasoning behind the final approaches. Finally, Section 1.5 outlines the overall structure of this thesis.

1.1 Background

The topic of this thesis is the development of Hyper-Heuristics (HHs) for the generation of perturbation operators to be used in search heuristics. To understand what this means, let us first explore and define the concepts treated in the pirate analogy in more depth.

1.1.1 Combinatorial Optimization

An important and active area of research in computer science and mathematics is that of Combinatorial Optimization (CO) [32]. CO problems are defined by a discrete set of solutions (or states) S to try, a set of constraints we have to follow ($g(x) > b$) and an objective function $f(x)$ which we want to either minimize or maximize, summarized as

$$\begin{array}{ll} \text{minimize/maximize} & f(x) \\ \text{subject to:} & g(x) \geq b \\ & x \in S \end{array} .$$

In the pirate analogy, the state space corresponds to all coordinates on the map, the constraints correspond to the border of the island and the objective value of a state corresponds corresponds the height of a location on the island.

Many practical questions can be reformulated as CO problems. An example of this is the Travelling Salesman Problem (TSP), which involves finding the shortest possible route that visits a given set of cities exactly once and returns to the origin city [44, 2]. The Vehicle Routing Problem (VRP) extends TSP by involving multiple vehicles that must deliver goods to various locations, aiming to minimize the total distance travelled or the cost incurred while satisfying constraints like vehicle capacity and delivery time windows [57]. Scheduling problems are another group of common CO problems. These problems require the allocation of resources over time to perform a collection of tasks. Examples are assigning employees to shifts or scheduling jobs on machines, to optimize specific criteria like minimizing total completion time or maximizing resource utilization [42, 35]. In this research, we will focus on the Trapdoor problem, which is discussed in detail in Section 2.1.

A naive approach to solving such problems would be to calculate the objective value of all possible solutions and pick the one with the optimal value. This method is known as brute-force search [10]. However, the number of potential solutions often grows exponentially with the size of the problem, a phenomenon known as combinatorial explosion [25]. As a result, evaluating every possible solution quickly becomes impractical for even moderately sized problems, as it would require an infeasible amount of computational time and resources.

In such cases, exact algorithms such as the Simplex method [11] can be applied to linear problems to find the global optimum or Heuristic algorithms can be used to approximate the optimum in a limited amount of time. Such heuristic algorithms are related to the topic of this thesis and will be discussed in more detail.

1.1.2 Heuristics

As stated, heuristics are algorithms that are not guaranteed to find the optimum but rather try to find a solution that is close (enough) to the optimum in a reasonable time [31]. A large variety of heuristics is used to solve CO problems.

Most have a similar structure: we start with a (generated) initial solution and make slight changes to this solution, often through the use of operator functions. An operator is a function that takes you from a current solution to one of its neighbours, often by changing a small part of the solution. We will refer to the part of the solution that is changed as the location of the application. These changes are accepted according to some acceptance criteria. After some ending criterion is met, the best-found solution is returned. We call algorithms of this form local optimization algorithms [27].

In the pirate analogy, the acceptance criterion was that we only accepted moves which didn't take us more than 20 centimeters down in one go, the initial solution corresponds to the starting place on the island, the operators correspond to steps or jumps and the stopping criterion to the sun setting.

There are many different variants of heuristics. For example, in hill-climber algorithms, we only accept moves which improve the objective function, in GRASP [20] several good solutions are combined to find new and better solutions and in Tabu-search [22] we keep a list of already visited solutions which we want to avoid to encourage exploration.

This thesis focuses on one heuristic to test the generated operators: Simulated Annealing (SA) [30]. This heuristic employs an alternative acceptance criterion in which the acceptance of a move is dependent on its effect on the objective value as well as how far along in the search the move takes place. This heuristic was selected for its widespread use and ability to balance exploration and exploitation. SA will be discussed in more detail in Section 2.2.2.

1.1.3 Hyper-Heuristics

In literature, heuristic algorithms are often designed and tuned for a specific type of problem. This means a new or slightly different problem may require an entirely new algorithm or different parameter settings of the same algorithm. Identifying the most suitable heuristic for a specific problem can be complex and time-consuming. This challenge has spurred the development of HHs, which are algorithms designed to automate the creation of heuristics for solving computationally demanding problems [5]. In the pirate analogy, the procedure used to learn to include jumps in the tactic can be seen as a HH.

More specifically, we can define a HH as a higher-level heuristic that manages a set of low-level heuristics, which searches for a method to solve a problem instead of a solution and does so using limited problem-specific information [6]. Comprehensive surveys of general HHs can be found in the work by Burke et al. [6] and more recently Sánchez et al. [48].

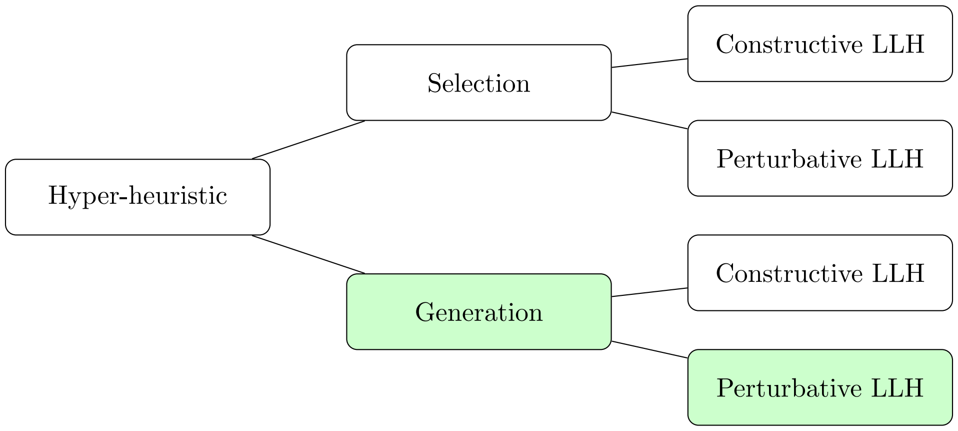


Figure 1.2: Classification of HHs [7]

As illustrated in Figure 1.2, HHs can be classified based on their objectives: either the selection or generation of Low-Level Heuristics (LLHs) or operators. These LLHs can be further categorized by their function: construction or perturbation [5]. Constructive operators build a solution from the ground up through successive applications, while perturbative operators introduce minor modifications to refine an existing solution. Perturbative LLHs are also referred to as neighbourhood structures.

An example of a selection HH for constructive LLHs is the Adaptive Large Neighbourhood search algorithm [45], which selects destroy and repair operators according to weights determined by their performance during search. Such destroy and repair operators decompose an existing solution and construct it again respectively.

One example of a HH for the selection of perturbative operators from the literature is the approach by [17]. They used reinforcement learning to guide the heuristic selection process in a HH framework. This approach involved different variations for the rewards, policy, and learning functions. They experimented with modelling the agents' states and actions in various ways to improve the selection of low-level heuristics for solving combinatorial optimization problems.

For the generation of LLHs (constructive or perturbative), evolutionary or grammatical methods are often employed. Since this thesis develops several HHs for the generation of perturbative LLHs, these techniques are discussed in more detail in Section 1.2.

1.1.4 Problem Statement

The primary objective of this research is to create and evaluate new HH approaches that can generate and prioritize effective perturbative operators for a combinatorial optimization (CO) problem, specifically the Trapdoor problem. Automating the generation of effective operators for heuristics removes the reliance on expert knowledge, allowing for easier and faster development of heuristics for solving CO problems, particularly those for which little expert knowledge is available.

We aim to generate these new operators through the recombination of an existing set of operators, referred to as *basis operators*. Such basis operators are relatively simple and should be easy to determine without in-depth knowledge of the problem. Instead of simply generating new operators through the composition of basis operators, we aim to include some additional surface-level information available on the problem, which can be used to further refine the composition of operators. This information is contained in the concept of a *relation* between the locations of application of two basis operators. This leads us to the concept of *chains*, which are composite operators with certain relations enforced between the locations of application of their sub-operators. In this research, we construct several chains which significantly improve the quality of search in SA when they are included as operators.

In addition, we aim to determine some mechanism for prioritizing effective chains, so that less computational time is wasted on testing ineffective chains through time-consuming SA runs. To this end, this study introduces three distinct HHs: Priority Tree Search (PTS), the Mining Hyper-Heuristic (M-HH), and the Population Mining Hyper-Heuristic (PM-HH). Each of these HHs relies solely on a surface-level understanding of the problem and does not require expert knowledge of effective operators or detailed information on how best to solve the problem. PTS exploits the tree-like structure of chains and prioritizes chains based on the effectiveness of their sub-chains. The other two HHs, M-HH and PM-HH, aim to use the wealth of information available in a single SA run to predict which chains are more likely to be effective.

1.2 Related Work

This section discusses previous research relevant to this thesis. It first explores studies highlighting the importance of operator selection within search heuristics, followed by an examination of research on the generation of such operators.

1.2.1 Effect of (Composite) Operators

In this work, we focus exclusively on generating perturbative operators, which is a critical component in the design of heuristic algorithms. The decision to concentrate on this aspect is motivated by the significant impact that the choice of operators can have on the algorithm's performance and speed, as illustrated intuitively in the pirate analogy. This claim is further supported by the related work discussed in this section.

A good choice of operator can make or break a heuristic algorithm. A great real-world example of such an effective operator can be found in the Lin–Kernighan heuristic [36], which is known as one of the most successful methods for generating optimal or near-optimal solutions for the Symmetric Travelling Salesman problem [24]. This heuristic's strength lies in its sophisticated use of the λ-opt operator, which is known for its high performance [19]. The λ-opt operator functions by cutting an existing tour into segments by removing λ links between cities and then recombining these segments in a new way to form a potentially shorter tour. The challenge with the λ-opt operator is the need to specify λ in advance, which can be limiting. The Lin-Kernighan heuristic overcomes this limitation by dynamically adjusting the value of λ. At each iteration, the heuristic evaluates whether increasing the value of λ could result in a shorter tour, thereby continuously optimizing the tour length [24]. This dynamic adjustment makes the Lin-Kernighan heuristic exceptionally effective, illustrating the critical role that adaptive and well-designed operators play in the success of heuristic algorithms.

Many other works exist outlining the difference in the performance of different operators for different instances. The necessity for different operators per type of problem instance was shown by Burke, Hyde, and Kendall [4] for different bin packing problems. Lü, Hao, and Glover [38] evaluated the performance of different neighbourhoods for the curriculum-based course timetabling problem. They found that good neighbourhoods can significantly improve the quality of search. In addition, it is found that the combination of neighbourhoods can lead to further improvements. The potential use of combining operators is also shown in the work by Di Gaspero and Schaerf [16] for problems in the timetabling family. However, the findings from this work suggest that the positive effect of such a composite operator may be negatively impacted by the size of the neighbourhood induced by it.

1.2.2 Generation of Perturbative LLHs

Generally, the generation of perturbative operators has been achieved through the recombination or configuration of existing operators or components thereof [6]. However, the domain is also considered to be under-researched [40, 41]. This section briefly discusses the previous research into the generation of perturbative operators.

Several similarities can be observed in the works discussed below. Firstly, the most frequently employed approaches are Genetic Programming (GP) and Grammatical Evolution (GE) [6, 40]. These techniques, inspired by natural selection, evolve sets of effective operators. The process involves assessing the fitness of evolved operators, where those with higher fitness are used to generate new operators through slight alterations or combinations. While these evolutionary methods have been successful in discovering effective operators, it is crucial to note that they are highly time-consuming. Depending on the complexity of the problem instances and the parameters used, the HHs discussed below may take several hours to several days to identify a suitable operator.

Secondly, given the typically large number of fitness evaluations required, most studies determine the fitness of operators through limited testing to ensure computational efficiency. Fitness is often evaluated by applying the operator a fixed number of times within a simple search heuristic. When the full run of an algorithm is considered, it is usually for relatively small problems or involves known optimal solutions, which may not be applicable to more general problems.

Lastly, these studies predominantly use relatively simple search heuristics, such as Hill-Climbing, both to determine the fitness of operators and to test the operators generated. This may limit the applicability of the discovered operators in more complex but effective search heuristics such as SA.

1.2.2.1 Genetic programming

Genetic Programming (GP), first established by Koza [33], is a technique for evolving computer programs, based on the concept of natural selection, and works as follows. A population of different programs is kept and their fitness is evaluated according to some objective. In every iteration, individuals (programs) in the population with higher fitness are used to generate the next generation of programs through the application of GP operators such as mutation, cross-over and replication. This iterative process of selection, crossover, and mutation continues over many generations. Each generation ideally improves the overall fitness of the population, gradually evolving programs that are better suited to the task at hand. This method allows GP to automatically discover programs that perform well, even for complex and poorly understood problems. A more detailed overview of genetic algorithm techniques can be found in the work by Katoch, Chauhan, and Kumar [29]. Several studies used GP to evolve perturbative LLHs.

Fukunaga [21] proposes the CLASS system. CLASS is a Genetic Programming based HH which is employed to develop local search operators for the NP-Complete Boolean Satisfiability problem (SAT) [9] by combining components from established human-derived heuristics such as GSAT [49], Novelty [39], and WalkSAT [50]. Instead of standard GP operators, CLASS uses five conditional branching rules for combining heuristics. The individuals are evaluated first on smaller problems and, if they can solve these instances consistently, on larger problems as well. The exact fitness function is based on the mean flips necessary to achieve success. Local search algorithms employing these operators were shown to be very competitive and even outperformed state-of-the-art algorithms.

Similarly, Bader-El-Den and Poli [3] utilized GP to generate disposable local search operators for SAT problems. This approach incorporated elements from human-derived perturbative heuristics, utilizing a specific grammar to orchestrate the combination of these components effectively to evolve LLHs. The found operators are then applied in a Stochastic Local search heuristic. Notably, some of the generated individuals corresponded to well-known LLHs, such as GSAT. The discovered LLHs are competitive with the performance of other evolutionary LLHs, as well as human-crafted ones. Only the LLHs generated in the work of Fukunaga [21] perform slightly better, but these are also slower to apply.

1.2.2.2 Grammatical evolution

Another approach to the generation of perturbative heuristics is through Grammatical Evolution (GE)[46]. Unlike traditional Genetic Programming (GP), GE uses a genotype-to-phenotype mapping process. This means that solutions are represented as variable-length bit strings (genotypes) which are then translated into executable programs (phenotypes) using a predefined grammar, typically Backus-Naur Form (BNF). Similar to GP, GE involves selection, crossover, and mutation applied to genotypes, but fitness is determined by evaluating the phenotypes generated from genotypes. A more detailed discussion of GE can be found in the work by Ryan, Collins, and Neill [46].

Sabar et al. [47] proposes a GE framework that takes several heuristics components (acceptance criteria, neighbourhood structures and neighbourhood combinations) as input and generates a local search template by appropriately combining these components. The technique also employs an adaptive memory mechanism, which contains a set of high-quality, diverse solutions. The fitness of individuals in the population is determined by (1) a problem-specific cost function and (2) based on the percentage of improvement of object function after the application of the operator on randomly generated solutions. The algorithm performs well on challenging problems from the Timetabling domain and the Capacitated Vehicle Routing domain with the same parameter settings and is competitive with the state-of-the-art as well as bespoke methods (at that time). This shows its generalizability across different problems.

Burke, Hyde, and Kendall [4] use GE to generate neighbourhood move operators, which can obtain optimal, or close to optimal solutions to the 1-D bin packing problem in a relatively small number of iterations. The fitness of an individual is determined by applying the operator several times on a problem instance. Only improving moves are accepted and the final solution is converted to a fitness by a specific bin-packing fitness function. The generated operators were tested within a hill-climbing heuristic. These evolved operators consistently outperformed human-designed constructive operators across various problem instances. The results also demonstrated that different bin packing problems require distinct neighbourhood operators, as operators effective for one problem instance did not necessarily perform well on others.

Stone, Hart, and Paechter [54] employ GE to generate LLHs to be applied in Iterated Local Search [37], for the Travelling Salesman Problem (TSP) and the Multi-Dimensional Knapsack Problem (MKP). The HH is applicable to the subset of problems that can be represented as an ordering problem. Notably, the technique relies on domain-specific LLHs. The fitness of individuals in the population is determined by applying the LLHs several times (1000 iterations for TSP and 2500 for MKP) in a hill-climber algorithm for 5 instances and taking the average. The found LLHs are shown to outperform human-designed operators for TSP (notably the 2-opt operator) and to be competitive for MKP.

More recently, Mweshi and Pillay [40] employed a grammatical evolution approach for generating different types of perturbative heuristics from scratch (a set of basic actions and components of the solution). The LLHs are applied in a hill-climber framework and an accept-all moves framework. The fitness of individuals is determined by applying the heuristic to a randomly generated solution. The technique is applied to Timetabling problems, the Capacitated Vehicle Routing problem and SAT problems. The results are competitive with those obtained by generation perturbative HHs and get relatively close to the best-known solutions.

1.2.3 Our contributions

Although the importance of operators within heuristics has been clearly demonstrated, surprisingly little research has been done into their automatic generation across domains. The main body of work which leverages the effect operators can have on performance focuses on the selection of operators rather than their generation [6]. Studies focusing on the generation of LLHs do so using Evolutionary Computing, which is an effective but time-consuming technique. In many cases, the decision of which operators to include still rests heavily on expert knowledge, which may not exist extensively enough for many specific problems.

This thesis aims to advance the field of HHs for combinatorial optimization by developing new methods for generating perturbative LLHs. The primary contributions of this work can be summarized as follows:

- We introduce a novel chain structure that enables the creation of new operators through the composition of simpler operators while limiting the size of the neighbourhood induced by it. Chains are built up from simpler basis operators and relations, which are expected to be relatively easy to deduce for a new problem. By utilizing this chain structure, we define several composite operators that significantly enhance the performance of SA for the Trapdoor Problem.
- Building on the chain structure, we develop a tree-based HH that systematically explores all possible chains up to a specified length. This method provides a comprehensive search strategy that ensures no potentially beneficial chains are overlooked.
- We develop two mining-based HHs that discover chains by leveraging information gathered during the SA search process, unlike traditional approaches that rely solely on end-of-search information. These mining-based techniques use intermediate feedback to guide the search for effective operator chains, meaning more information is available through fewer runs of SA. Unlike the discussed methods for generating perturbative operators, the HHs developed in this thesis do not rely on genetic programming.
- Finally, to our knowledge, this thesis presents the first test of the generated operators in SA, as opposed to more simple search heuristics.

1.3 Relevance

This section is dedicated to outlining the relevance for this research.

1.3.1 CQM Case Study

The direction of this research is partly inspired by a real-world problem encountered by Consultants in Quantitative Methods (CQM) in one of their applications, RitOpt 4.

RitOpt 4 is a SA-based heuristic algorithm designed to solve various forms of Vehicle Routing Problems (VRP). It is engineered to handle problems with a variable number of vehicles, different time windows, post-unloading requirements, and varying scheduling duration. The algorithm uses many operators since the RitOpt 4 application is designed to be generally applicable. During the search process, operators are selected based on probabilities derived from a distribution, which was manually tuned by the algorithm designer's intuition.

However, it became evident during running that the acceptance rates of different operators could vary significantly across different problem instances. Some operators frequently rendered solutions infeasible, causing dramatic drops in the objective function due to penalties incorporated in the objective function. Moreover, the beneficial effects of certain operators were only realized when used in conjunction with others, highlighting the potential for improved performance through operator combinations. This insight led to the concept of generating complex operators by chaining existing ones together.

The specific challenge posed by CQM was: "*How can Reinforcement Learning techniques be employed to generate new operators (by combining multiple existing operators) that enhance the quality of the SA search?*"

In this thesis, we have addressed this question by developing various HH approaches that rely on this chain structure. These approaches serve as a proof of concept for mining-based techniques for the development of new operators in an existing implementation of SA, such as the RitOpt 4 application.

1.3.2 Broad Applicability

The techniques discussed in this thesis contribute to the field of HHs by introducing and validating, to a limited degree, an alternative approach to the generation of perturbative operators. If further developed, these techniques could have significant applicability, as they are relevant to any search heuristic that employs perturbative operators to solve combinatorial optimization problems.

Combinatorial optimization is a significant area of research with numerous real-world applications, including scheduling, routing, and resource allocation. The techniques developed in this thesis are designed to enhance the performance of heuristic algorithms by improving the generation of perturbative operators. As emphasized in Section 1.2, the choice of operators is crucial to the performance of heuristic algorithms. Therefore, effective techniques for generating operators can substantially impact the ability of search heuristics to solve combinatorial optimization problems. By automating the creation of effective composite operators, these methods can be applied to a wide variety of heuristic algorithms, potentially improving their efficiency and effectiveness across different domains.

The positive effects of such HHs extend beyond merely saving time for algorithm designers. They enable faster and more robust optimization without the need for extensive manual tuning or domain-specific expertise. This allows researchers and practitioners to focus on other critical aspects of algorithm development and application. Additionally, the ability to automatically generate effective operators reduces the risk of dismissing potentially valuable techniques due to suboptimal operator choices.

Additionally, these techniques can help bridge the gap between domain-specific algorithms and more general approaches. By eliminating the necessity for manually designing operators for each new problem domain, these HHs make it easier to apply algorithms to a diverse range of problems without extensive customization.

This broad applicability highlights the importance and impact of this research in advancing the field of optimization.

1.4 The journey

Over the course of this thesis, various approaches were considered to address the original question posed by CQM. Given the limited existing research on this topic, multiple options needed to be explored and evaluated. Often, potential directions were investigated but ultimately dismissed for various reasons. These exploratory sidesteps occupied a significant portion of the research time, leaving limited time for refining the techniques that were eventually settled on. Nonetheless, each dismissed approach also provided valuable insights into the problem, shaping the final HHs developed. This section briefly discusses these initial attempts chronologically.

1.4.1 Q Learning

The initial approach to solving the research problem using Reinforcement Learning (RL) involved investigating Q-learning due to its prominence within RL techniques [8]. This decision was based on prior research indicating successful applications of Q-learning in heuristic contexts, such as determining start solutions and operator selection.

Q-learning is a model-free reinforcement learning algorithm that seeks to learn the value of an action in a particular state. The goal is to develop a policy that tells an agent what action to take under what circumstances to maximize some notion of cumulative reward. The core idea behind Q-learning is to iteratively update the Q-values, which represent the expected utility of taking a given action in a given state, and following the optimal policy thereafter.

The Q-learning algorithm updates the Q-values using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

where α is the learning rate, γ is the discount factor, r is the immediate reward, and s' is the state reached after taking action a.

A critical aspect of Q-learning is balancing exploration and exploitation, often achieved using an epsilon-greedy strategy, which allows the agent to greedily exploit promising policies as well as take random steps to explore other policies.

1.4.1.1 Literature on Q-Learning in Hyper-Heuristics

Q-learning has been extensively utilized in heuristic optimization for various purposes. For instance, in the realm of operator selection, Q-learning has been applied to dynamically adapt the search strategy based on the state of the search and past actions, as highlighted by Burke et al. [5]. Various studies have further explored the adaptive selection of operators using Q-learning within HHs. For example, Karimi-Mamaghan et al. [28] proposed a Q-learning-based Iterated Local Search (Q-ILS) method, where Q-values were used to adaptively choose between different local search operators based on their historical performance.

Additionally, Q-learning has been integrated with other heuristic methods to solve combinatorial optimization problems. Delarue, Anderson, and Tjandraatmadja [14] investigated using Q-learning to optimize operator selection in vehicle routing problems (VRPs), demonstrating significant improvements in solution quality. This integration showcases the versatility of Q-learning in enhancing heuristic algorithms across various problem domains.

The concept of learning macro-actions in reinforcement learning has also been explored, where sequences of actions (operators) are learned to optimize performance. Randlov [43] discussed how Q-learning could be extended to learn these macro-actions, significantly improving the efficiency of the learning process. By focusing on sequences of actions, the algorithm can capture more complex behaviours and interactions, leading to better performance in solving optimization problems.

1.4.1.2 Challenges

Although Q-learning has demonstrated considerable potential in optimizing heuristics, several inherent challenges make it unsuitable for our specific problem.

One of the issues with Q-learning is the exponential growth of the state space when considering a longer history of past actions. As the state space expands, the model becomes increasingly complex, requiring extensive computational resources, resulting in very slow convergence. This slow convergence is particularly problematic as we expect the best composite operators to consist of more operations.

Additionally, Q-learning's heavy reliance on the reward function and various parameters poses significant challenges. Tuning the reward function to balance short-term and long-term rewards is difficult. SA, the heuristic in which we test operators and apply the Q-learner, benefits from accepting suboptimal solutions early on to escape local minima. This characteristic of SA conflicts with Q-learning's emphasis on short-term rewards, making it challenging to align the learning process with our long-term optimization goals.

Furthermore, Q-learning's high dependence on chance for identifying effective operator sequences introduces significant inefficiencies. For Q-learning to identify beneficial operator sequences, the correct sequence of operators must appear in the right order and be applied to the appropriate locations within the state of the solution. This dependence on chance is particularly problematic for longer sequences of operators, where the likelihood of encountering the positive signals needed for effective learning is extremely low. The model also struggles to account for the specific locations of application and the state of the solution, further complicating the identification of effective sequences.

We conclude that Q-learning is not a suitable technique for our problem. We require a technique that can efficiently identify composite operators involving more operators, reduce the significant role of chance in finding effective composite operators, and operate without extensive parameter tuning. Moreover, the discovered operators must be effective in the long term, improving the final results of a whole heuristics search, not just after a few iterations.

1.4.2 The introduction of chains

Within Q-learning we implicitly specified the exact shape we allowed the generated operators to have. Namely, they were created by combining the sequential applications of operators into one operator. This composition of operators as a method to create new operators has inherent drawbacks, primarily due to the increase in neighbourhood size it causes, as discussed in Section 1.2.1. For operators with a larger neighbourhood size, it becomes increasingly challenging to evaluate the quality of the operator. For example, an operator that randomly shuffles the solution could potentially send the solution to any other possible state, good or bad, making it hard to gauge its effectiveness. To address this issue, we introduce the concept of *chains*.

Chains are composite operators in which the location of application for sub-operators is restricted, preventing the neighbourhood size from growing uncontrollably with each composition. By specifying the locations where each sub-operator within a composite operator is applied, we maintain a manageable neighbourhood size. This approach ensures that the composite operators remain specific and related to the particular areas they are intended to improve.

Moreover, the concept of chains aids in addressing another significant issue: the high dependency on chance in identifying effective operators. When operators are simply composed without any structure, the probability of their random combination yielding beneficial results reliably is low. This is particularly problematic for longer sequences of operators, where the likelihood of achieving a successful composite operator by chance diminishes further. Chains mitigate this problem by providing a structured way to combine operators, thereby increasing the chances of discovering effective composite operators systematically.

Chains play a key role in the HHs developed in this thesis and will be discussed in more detail in Section 2.3. By introducing the chain structure, we have essentially defined *how* we generate operators. The relevant questions now remain: How do we determine which chains are the most effective, and given limited computational resources, which chains should be prioritized for testing?

In Section 3.1, we will detail a method for evaluating the effectiveness of a chain operator by analyzing its impact on the quality of the solutions returned by a search that includes the operator. This evaluation will be referred to as the *performance* of a chain. The different HHs discussed in this thesis—Priority Tree Search (PTS), Mining Hyper-Heuristic (M-HH), and Population Mining Hyper-Heuristic (PM-HH)—are designed to address the question of the optimal order in which chains should be tested for their performance.

1.4.3 Tree Search

Given the introduction of chains, we can reformulate the problem as a tree search. This approach allows us to systematically explore the potential combinations of basis operators. Essentially, each chain of operators can be visualized as a tree where each branching corresponds to a choice of the next basis operator. In this structure, each node represents an operator and its application, while each path from the root to a leaf node represents a complete chain of operators. The depth of the tree corresponds to the length of the chain.

A key challenge in this approach is the combinatorial explosion of possible chains as the tree expands, either because we include longer chains or a larger set of potential building blocks for the chains. If we assume that chains exist whose inclusion significantly improves the quality of search, it's not guaranteed that a subchain of this optimal chain will also perform well. Moreover, determining the fitness of each chain is computationally expensive since it involves running the search algorithm multiple times with each chain.

1.4.3.1 Priority Tree Search

In an attempt to explore this tree efficiently, we introduce the Priority Tree Search (PTS) algorithm. PTS is designed to search the tree based on the performance of the chains within an SA search. It prioritizes the exploration (i.e. extension) of chains that show promise by evaluating their performance and expanding the most promising nodes first. This method helps in managing the combinatorial explosion by focusing computational resources on the parts of the tree that are more likely to contain effective chains. This approach relies on the assumption that sub-chains of effective chains are also effective.

However, in testing the algorithm, we find that although PTS eventually discovers all chains, it does not efficiently prioritize chains that have a positive effect on the search. For a detailed discussion of the algorithm and the results of testing please refer to Sections 3.2 and 5.2. We found that the underlying assumption that sub-chains of effective chains are also effective does not always hold. In addition to this, we find that in some cases, PTS does not perform better than randomly selecting and testing chains. This illustrates the difficulty of discovering effective chains efficiently, even when we limit the set of all operators to the set of chains (defined for some set of basis operators and relations between them).

1.4.3.2 (Monte-Carlo) tree search

To counteract the combinatorial explosion in the tree search, we considered the application of Monte Carlo Tree Search (MCTS). MCTS is a heuristic search algorithm that has been successful in large search spaces, particularly those involving consecutive decisions and opponent agents, such as in game playing. The algorithm helps determine which subtrees should be explored further. It does so by training two neural networks: (1) a value network which determines the value of a leaf for the player and (2) a fast rollout policy used to complete a game through self-play. One of the most notable applications of MCTS is in AlphaGo, the computer program developed by DeepMind that defeated human champions in the game of Go [51].

However, several significant differences between our problem and typical applications of MCTS pose challenges. Firstly, our problem lacks an opponent, which is a fundamental aspect of traditional MCTS applications. Some research has attempted to address this issue using techniques like ranked rewards to simulate competition [34], but this approach may not fully capture the nuances of our problem.

A more critical challenge is that the rewards in our problem are highly dependent on the state of the solution and can vary significantly due to the inherent stochasticity of the search heuristic. While the introduction of chains has reduced the role of chance, it remains an integral part of search heuristics like SA. To account for the probabilistic nature of the search, a complete tree would also need to include the state of the solution at each step. This requirement would dramatically increase the size of the tree, leading to concerns about convergence and computational complexity. Given these considerations, we elected not to employ MCTS for this problem.

1.4.4 Mining-Based Hyper-Heuristics

PTS determines which chains to expand and test based on their performance in SA. This performance is evaluated solely by considering the final objective value returned by the SA run. However, a single SA run contains significantly more information than just the final solution. By leveraging this additional information, we can enhance the effectiveness of our HH approach and potentially reduce the computational time required. This is the fundamental idea behind the two mining-based approaches developed in this thesis: the Mining Hyper-Heuristic (M-HH) and the Population Mining Hyper-Heuristic (PM-HH).

1.4.4.1 M-HH

M-HH analyses the set of accepted moves as they appear in an SA search. For shorter sections of moves where desirable behaviour is observed (i.e. a dramatic increase in objective value), it determines which chains appear more often and thus are likely to be the cause of the desirable behaviour. Such chains are assigned a higher *support* value. By ordering chains from high to low support value, an ordering of chains is induced, which can be used to determine which chains to test within SA first.

When testing the HH, we find that support value is a good predictor of chain performance and that the ordering induced by support value effectively prioritizes high-performance chains. However, several chains do not appear in the ordering as they are not encountered often enough in the analyzed sections. Detailed implementation and results of the M-HH are discussed in Sections 3.3 and 5.3.

1.4.4.2 PM-HH

The PM-HH builds upon the M-HH algorithm by incorporating several high-support chains discovered through M-HH directly into SA as operators and then repeating the support calculation process. This technique borrows some ideas (namely a population which updates every generation) from the Genetic Programming techniques discussed in Section 1.2.2.1. By continually refining the set of promising chains and incorporating them into the SA runs, PM-HH ensures that more information is available on high-support chains and their variations, which can then be prioritized for testing performance. Testing the PM-HH reveals that while it discovers a larger number of high-performance chains, the relationship between support value and performance is less consistent compared to M-HH. This inconsistency is likely due to the current definition of desirable behaviour, which may include moves that result in back-and-forth behavior rather than continuous improvement. The PM-HH method is detailed in Section 3.4 and its performance is evaluated in Section 5.4.

1.5 Thesis Organization

In this chapter, we have established the foundation by outlining the research problem, objectives, and significance. Additionally, we have provided an overview of the research trajectory. The rest of this thesis is structured as follows:

In Chapter 2 we detail the background theory necessary to understand the developed techniques. In this chapter we will discuss the Trapdoor problem, SA and the chain structure.

Chapter 3 details the design and implementation of the HH approaches developed in this thesis. It introduces PTS, M-HH, and PM-HH. Details on the method used for determining the performance of chain are also discussed in this chapter.

Chapter 4 presents preliminary experiments conducted to determine necessary parameter settings. This chapter also evaluates the performance of several chains for the Trapdoor problem, setting the stage for more in-depth analysis in later chapters.

In Chapter 5 we will discuss several experiments conducted to determine the effectiveness of the different HHs developed in terms of the ordering of chains they return. Each HH will be discussed individually, and be compared to the different HHs.

Finally, Chapter 6 summarizes the findings of this research and offers suggestions for future studies. This chapter reflects on the strengths and limitations of the proposed methods and outlines potential directions for further exploration in the field of HHs.

Chapter 2

Theory

As stated, this research aims to develop techniques for the generation of perturbative operators, which are used in search heuristics applied to solve combinatorial optimization problems. This chapter provides a theoretical background on several key aspects that remain consistent throughout the study: the problem we aim to solve, the heuristic employed, and the format used to generate new operators.

To illustrate the potential of the developed Hyper-Heuristics (HHs), we test them on a well-understood problem, the Trapdoor Problem, using a common search heuristic, Simulated Annealing (SA). The Trapdoor Problem provides a suitable testbed due to its simplicity and the ease with which we can deduce the optimal operators, yet it remains sufficiently challenging to solve optimally without the right operators. SA is chosen for its widespread use and its inherent mechanism for accepting non-improving moves, which aligns well with the principles of our HHs.

All HHs developed in this research generate operators with a specific structure. We will refer to these operators as *chains*. This structure makes it possible to create composite operators from simpler, more basic operators. These chains are designed to enhance the search process by combining multiple simple moves into a single, more powerful operator. It is expected that in many cases, determining the building blocks for such chains is easier than creating a new operator from scratch. This structure was created with the ability to escape local optima in mind.

This chapter is structured as follows: We begin with a detailed description of the Trapdoor Problem in Section 2.1, discussing the basics of the problem, the associated objective function, and the motivation for its choice.

Following this, we delve into the principles of SA in Section 2.2.2, explaining its fundamental mechanics, acceptance criteria, and the role of temperature in the search process. We highlight why SA is a fitting choice for this research and how it benefits from the application of complex operators generated through HHs.

Finally, after some intuition has been created for the Trapdoor problem and SA, we introduce the concept of *chains* in Section 2.3. In this section, we define how these chains are constructed from basic operators and connected by *relations*, providing several examples of chains in the context of the Trapdoor Problem. We also explore the rationale and underlying ideas that inspired the definition of this structure.

2.1 Trapdoor

In this section, the Trapdoor Problem is introduced in more detail. We elected to test the developed HHs on the Trapdoor problem for several reasons. For one, for the Trapdoor Problem, the effective operators can be easily deduced using knowledge of the underlying problem but can be hard to discover without this information. This means that we already know which operators we aim to generate, and can compare the effectiveness of different HHs based on the discovery of these operators. More details on the specific operators which we hope to discover can be found in Section 2.3.

In addition, the problem is sufficiently difficult, as it contains several local optima, depending on the choice of operators. This means that a proper choice of operators is necessary to solve the problem efficiently. For a bad choice of operators, the Trapdoor problem has many locally maximal solutions, which can be hard to escape.

Finally, the relative difficulty of the problem can be easily altered by changing the size of the problem. More details on how the problem is solved by the SA search heuristic can be found in Section 2.2.2.

2.1.1 Details of the problem

In the Trapdoor problem, there are R rooms or rows, and in each room, there are C bits. Which position the bit is in we call the column. Based on the value of the bits we can calculate the objective function. Generally, the value of the objective function is the sum of all bits. However, for each room which contains only zeroes we add $C + B$ for some bonus $B > 0$ to the objective function. The Trapdoor Problem is a maximization problem, meaning we aim to find the solution with the highest possible objective value associated. The number of possible solutions for a Trapdoor problem of size $C \times B$ is equal to $2^{C \cdot B}$, since the problem contains $C \cdot B$ bits and each bit can have two values.

Definition 1. *Let $X \in \chi^n = \{0,1\}^{R \times C}$ be a solution to the Trapdoor problem, Let $(X_i)_{i \in [R]}$ be the collection of rooms in X. The objective value of a room X_i is given by*

$$f_r(X_i) = \begin{cases} C + B & \text{if } \sum_{j=1}^{C} X_{ij} = 0 \\ \sum_{j=1}^{C} X_{ij} & \text{otherwise,} \end{cases}$$

and the objective value of X is given by

$$f(X) = \sum_{i=1}^{R} f_r(X_i).$$

This means that in the optimal solution, all rooms contain only zeroes. However, all cases where a room contains only 1's is a local maximum as well. This means that, depending on your operator choice, the problem often cannot be solved with simple hill-climbing tactics.

In this research, we will work with a Trapdoor problem with room size $R = 6$ and column size $C = 6$. For such a problem there already exist more than 68 billion (2^{36}) possible solutions. Relatively small values were chosen such that a single SA run would not take too much time (less than a second), and still be able to achieve reasonable results (solutions in which some rooms contain only 0s). The bonus for a room with only zeros is set to $B = 4$. For the HHs developed the exact value of the bonus does not matter, as long as $B > 0$.

With these parameters, the objective value lies within the interval $[6, 60]$. Here the lowest objective value is achieved when there is only one 1 bit in every room, and the highest objective value is achieved when all rooms contain only 0s. If all bits in the solution are equal to 1 (a local maximum), then the objective value is equal to 36.

2.2 Heuristic

The HHs developed in this research generate operators which may be used within any search heuristic which uses operators. Throughout our research, we will focus on one specific search heuristic: Simulated Annealing (SA). The SA algorithm is not only used to test the effectiveness of these operators but also plays a role in their generation. More details on this can be found in Chapter 3.

We chose SA as our search heuristic for several reasons. Firstly, SA is the heuristic that inspired the original question from CQM. Secondly, SA is widely used and well-regarded in the field of optimization [13]. Thirdly, SA includes a built-in mechanism for encouraging explorative changes that do not immediately improve the objective function. The fact that these 'missteps' are allowed throughout the algorithm, makes it possible to learn from them, as in done in the mining-based HHs.

In this section, we delve into the specifics of SA. We start with an explanation of its mechanics, beginning with an overview of Hill-Climbing to provide context. Following this, we elaborate on key aspects of SA, including its acceptance criterion, parameters, neighbourhood structure, methods for incorporating constraints, and strategies for calculating the objective value.

2.2.1 Hill-Climbing

SA is most easily explained as an expansion of the classic random Hill-Climbing approach. Consider a problem P without constraints with solutions space S. A hill-climber algorithm starts with a random solution in the state space S. It then iteratively choose a solution similar to the current solution by changing a limited aspect of the solution (selecting one of its neighbours) and makes it the new current solution if this step improves the objective function. We will often refer to the change from one state to another as a *move*. In other words, the *acceptance criterion* for the move is that it must improve the objective function. These changes from move to move continue until some *termination criterion* is met. Some examples of termination criteria are: a maximum number of moves has been performed or no improvement has been found after a fixed number of moves. Algorithm 1 shows the pseudocode for the Hill-Climbing algorithm.

Algorithm 1: Hill-Climbing

```
Input  : A solution $s \in S$ (a solution in the solution space)
Output: An optimized solution $s$

Function LocalSearch($s$):
    repeat
        $s' \leftarrow$ Neighbor($s$)
        if $f(s') > f(s)$ then
            $s \leftarrow s'$;
    until $Termination\ criterion == True$;
    return $s$;
```

In this algorithm, S denotes the solution space, s is the current solution, $f(s)$ is the objective function to be maximized and the function `Neighbor`(s) generates a neighbouring solution of s.

A major pitfall of the Hill-Climbing algorithm is the fact that it cannot escape a local optimum. This problem is addressed in the SA algorithm.

2.2.2 Simulated Annealing

Simulated Annealing was first introduced in 1982 by Kirkpatrick, Gelatt, and Vecchi [30], and was inspired by the annealing process in metallurgy. This process involves heating a solid to its melting temperature and then cooling it slowly to achieve a state of minimal energy. If the initial temperature is sufficiently high and the cooling rate is sufficiently slow, it has been proven that the minimum energy state is always reached [13].

Similarly, the SA algorithm employs a temperature variable that decreases according to a predetermined cooling schedule. This temperature variable is used to determine how likely it is that a 'bad' move is accepted. SA thus expands on the Hill-Climbing framework by allowing not only improvements ($f(s') \geq f(s)$) but also occasional acceptance of non-improving moves ($f(s') < f(s)$). The probability that a worse move is accepted is determined not only by the current temperature but also by the size of the change in the objective value, $\Delta = f(s') - f(s)$.

The pseudocode for a basic version of SA as implemented for this research is shown in Algorithm 2. It uses an exponential decay schedule and a final temperature as the stopping criterion. There are many different possible versions of SA, including but not limited to SA with random restarts, alternative acceptance probabilities and alternative temperature schedules [18].

In this algorithm, S denotes the solution space, s is the current solution, f is the objective function to be maximized and the function `Neighbor`(s) generates a neighbouring solution of s. The temperature variable is T.

2.2.2.1 Acceptance Criterion

A key aspect that differentiates SA from basic Hill-Climbing algorithms is its acceptance criterion. In Hill-Climbers, typically only better solutions are accepted. In contrast, SA uses a probabilistic acceptance rule

Algorithm 2: Simulated Annealing

```
Input  : Initial solution $s \in S$,
         initial temperature $T_{init}$,
         end temperature $T_{end}$,
         cooling rate $\alpha$,
         number of iterations $n$,
         global operators $\Theta_g$
         operator probabilities $P$.
Output: The best solution found $best$.

Function Simulated_Annealing($s, T_{init}, T_{end}, \alpha, n, \Theta_g$):
    $best \leftarrow s$ ;
    Initialize $T \leftarrow T_{init}$ ;
    repeat
        for $k \leftarrow 1$ to $n$ do
            Select Neighbour function from $\Theta_g$ according to distribution $P$ ;   ▷ Select operator
            $s' \leftarrow$ Neighbour($s$) ;
            if $f(s') > f(s)$ then
                $s \leftarrow s'$ ;                                                    ▷ Accept improving move
            else
                $\Delta \leftarrow f(s') - f(s)$ ;
                $p \leftarrow \exp(\Delta/T)$ ;
                if $random(0,1) < p$ then
                    $s \leftarrow s'$ ;                                                ▷ Accept worse move
            if $f(s) > f(best)$ then
                $best \leftarrow s$ ;                                                  ▷ Update best solution found
            $T \leftarrow \alpha T$ ;                                                  ▷ Reduce temperature
    until $T < T_{end}$;
    return $best$ ;
```

that allows it to accept worse solutions under certain conditions, which helps it to escape local optima and explore the solution space more thoroughly.

The acceptance of a new state in SA, generated by applying an operator to the current state, involves calculating the change in the objective function value, denoted as $\Delta = f(s') - f(s)$. Without loss of generality, we assume we are working with a maximization problem. Then,

- If Δ is positive (indicating an improvement), the new state is always accepted.
- If Δ is negative (indicating a deterioration), the new state is accepted with a probability given by $e^{\frac{\Delta}{T}}$, where T is the current temperature. This probability decreases as the temperature falls, reducing the likelihood of non-improving moves as the system 'cools down'.

Figure 2.1 visualises how different values of T and Δ affect the probability of acceptance. Non-improving moves are more likely to be accepted at higher temperatures or when the decrease in the objective value is smaller.

The SA algorithm can generally be divided into two phases: exploration and exploitation. At the start of the algorithm, the temperature is high, and worsening moves are frequently accepted. This phase allows the algorithm to explore a larger portion of the solution space, reaching areas that a simple Hill-Climbing algorithm could not. The acceptance of suboptimal solutions prevents the algorithm from getting trapped in local optima early on, promoting a thorough search of the solution landscape.

As the temperature gradually decreases, the algorithm transitions into the exploitation phase. In this phase, only improving moves are accepted, similar to the behavior of a Hill-Climbing algorithm. This targeted

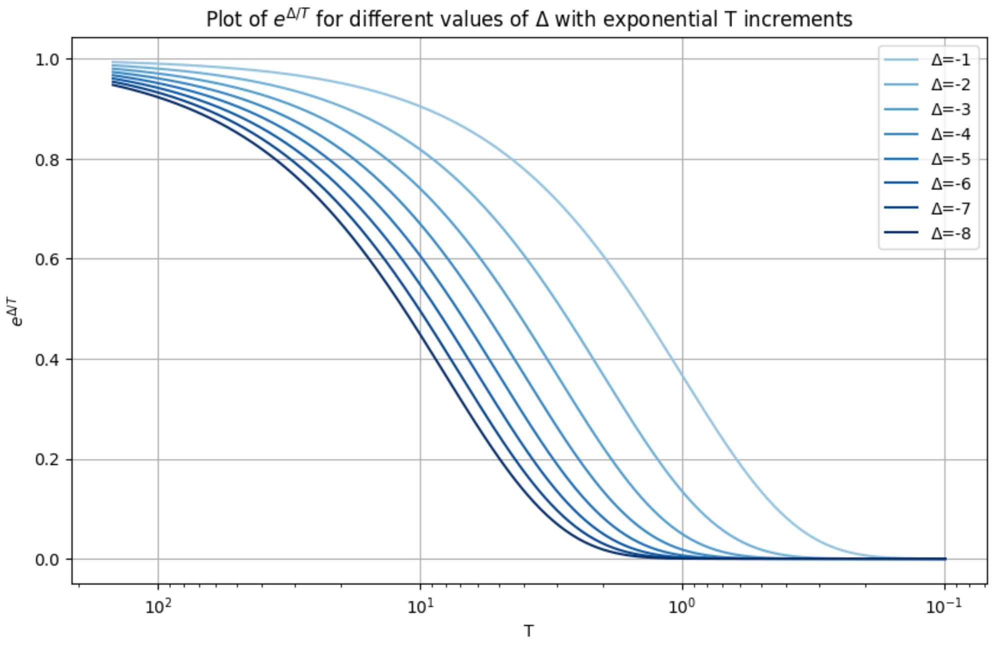


Figure 2.1: The probability of acceptance for different temperatures and difference in objective function Δ. Temperature decreases according to an exponential decay schedule.

search focuses on refining the current solution to find the locally optimal solution within the explored area. By narrowing the search, the algorithm hones in on the best possible solution in its vicinity.

These two phases enable SA to balance between searching widely for global optima and fine-tuning solutions to achieve local optimality.

2.2.2.2 Neighbourhood Structure

In SA, the potential next state is picked from one of the neighbours of the current state using the `Neighbour` function. This is a function that makes small changes to the current solution. We often refer to such functions as (neighbourhood) operators or Low-Level Heuristics. It is common for different types of operator functions to be used in a single implementation. In this implementation, we select an operator from the set of *global operators*, Θ_g, according to some distribution P. Each element of P represents the probability of selecting the corresponding operator from Θ_g.

The effectiveness of SA in solving different problems depends on the appropriate selection of operators, which is influenced by the problem's underlying structure and the shape of its solution space. Developing a suitable set of operators is the focus of this research, and the concept is further explored in Section 2.3. A fundamental requirement when choosing operators for SA is that it must be possible to reach all states from the initial state[13]. Failing to meet this requirement means the algorithm cannot fully explore the search space.

For the Trapdoor problem, a likely choice of operator is the bit flip operator, which randomly selects a bit and flips it from 0 to 1 or from 1 to 0 (`bitflip`). An alternative approach could be to include operators which always flip a bit to 1 or always flip a bit to 0 (`flip_to_1` and `flip_to_0`). More details on operators can be found in Section 2.3.

2.2.2.3 Parameters

To implement the algorithm, the following parameters need to be configured:

- **Initial Temperature, T_{init}**: This parameter sets the starting temperature of the SA algorithm. A higher initial temperature allows the algorithm to explore the solution space more freely, increasing the probability of escaping local minima early in the process. However, setting the temperature too high means time is wasted making random moves that do not contribute to finding a better final solution.

- **End Temperature, T_{end}**: This is the temperature at which the annealing process is terminated. When the system's temperature reaches or falls below this threshold, the algorithm stops and the best solution found thus far is returned.
- **Cooling rate**, $0 < \alpha < 1$: This parameter determines how fast the temperature decreases throughout the algorithm. For higher values of α the temperature decreases slower.
- **Number of iterations per temperature**, n: This parameter determines the number of moves the algorithm may try before a temperature update occurs. This includes both accepted and rejected moves. Increasing n means the T decreases slower, allowing for more exploration.
- **Global operators**, Θ_g: This is the set of operator functions which may be used to change the solution.
- **Operator probabilities**, P: This parameter determines how likely it is that each operator in Θ_g is selected.

In our implementation we have set $T_{\text{init}} = 150$, $T_{\text{end}} = 0.1$, $\alpha = 0.99$ and $n = 30$. The start solution is randomized by initializing each bit as 0 or 1 with equal probability. The set of global operators and the associated probabilities varies.

These values were chosen so that the SA algorithm with only the `bitflip` operator finds a reasonable solution (some rooms contain only zeros) but rarely finds the optimal solution. In Section 4.1, we find that SA with these parameter settings finds a solution of an average objective value of 50, meaning that around 3.5 rooms contain only 0s. This way a single run of SA displays some desired behaviours which we can reinforce, but there is also still room for improvement. This is meant to simulate a real-world SA algorithm for which parameter settings have already been optimized for the given operators and time constraints.

2.2.2.4 Incorporating Constraints

Although constraints are not explicitly addressed in the standard SA algorithm, this does not mean it cannot be used for solving constrained optimization problems. The simplest method to incorporate constraints is to use a feasibility-preserving neighbourhood structure that generates or considers only those neighbours that comply with the constraints [13]. This ensures that every candidate solution explored by the algorithm is valid, simplifying the optimization process. Another common approach for handling constraints within SA is to incorporate a penalty element into the objective function [18, 55]. This penalty element assigns a high cost to solutions that violate the constraints, effectively discouraging the algorithm from considering infeasible solutions. Alternatively, repair mechanisms can be used to adjust infeasible solutions to make them feasible before evaluating their objective function values.

The Trapdoor problems has no constraints, so these methods are not necessarily for our implementation. However, it is important to highlight that the HHs developed in this research may also be applied to problem *with* constraints as well.

2.2.2.5 Calculating Objective Value

In the pseudocode, we show the objective value as being calculated completely anew for a new solution. However, in many implementations of SA or other search heuristics, the objective value of the new solution is calculated incrementally by calculating the difference in objective value for the part of the solution that is changed [13]. This way, the objective value is not recalculated over and over again for parts of the solution that remain unchanged, speeding up the algorithm.

Similarly, in our implementation for the Trapdoor problem, the difference in objective value, Δ, is calculated by determining the difference in objective value for the specific room(s) affected by an operator. This means that if only room i is affected by an operator, then we can calculate the difference in objective value as follows:

$$\Delta = f_r(s_i') - f_r(s_i).$$

Here f_r is the objective value function of a room as defined in Definition 1, s_i is the room i before the change and s'_i is the room i after the change. If multiple rooms are affected by an operator, then we sum over the differences in objective value for each of the rooms.

2.3 Our Goal: Finding Composite Operators

As stated, the goal of this research is to generate new, more complex operators to be used in SA. The shape that these operators may take plays a crucial role in the way they are generated.

In this research, we generate such operators by chaining operators together, i.e. by applying them successively. This section outlines and motivates the exact format we allow these complex operators to have. All concepts are explained generally for a problem P without constraints, with solution space S. As per Section 2.2.2.4, a constrained problem can be reduced to such a problem.

We start by motivating the choice of the structure by considering which properties we expect an effective operator to have. Next, we go into more detail on the exact definitions of an operator. Finally, we define the concept of *chains*. All defined concepts are illustrated further with examples from the Trapdoor problem.

2.3.1 Desired Properties of (Composite) Operators

Since the goal of this research is to discover operators that improve the annealing process, we discuss some relevant characteristics of search which we expect to correlate with higher performance. For the SAT problem, three useful measures have been previously defined [21, 52]: depth, mobility and coverage. In addition to these, we discuss a possible fourth property: smoothness. Since a set of operators induces a search, these features can also be used to discuss the qualities of specific operators or a combination of operators.

Depth, Mobility and Coverage

Using the objective function to decide whether to accept a neighbour during a search implicitly assumes that good solutions are close together. Under this assumption, it is natural to assume that spending more time in regions with high objective functions means that we are more likely to find a good solution. To this end, *depth* measures the extent to which a search algorithm explores states with good objective function values. The depth of a search is calculated by taking the average objective function per step (after ignoring a fixed number of initial steps).

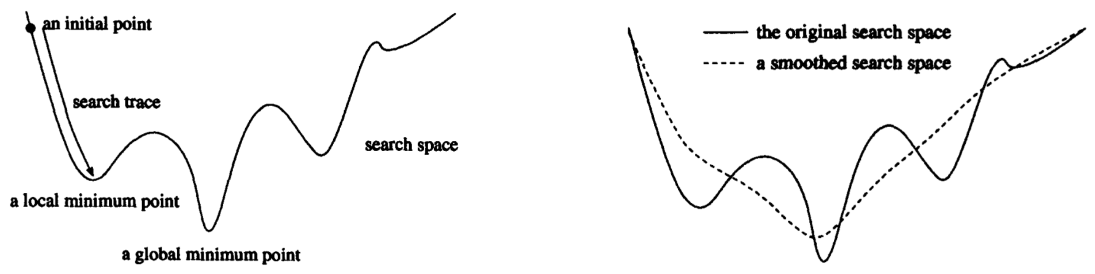
Clearly, depth alone is not a good indicator of the quality of an operator. A successful search does not only focus on a single (local) maximum, but also explores other regions. This ability to explore and accept worse solutions is what makes SA such a powerful tool. To this end, *mobility* measures the tendency to cover ground in the search space. It can be summarised as the average distance covered in a given number of steps. This value is often normalised by dividing by the maximum step size.

If an operator has high mobility, it ensures that we do not simply sit in a place with a high objective function. However, since the mobility only takes the average within a certain number of steps, a search can get stuck in a larger circle of solutions and still have high mobility (and depth). To account for this, the *coverage* measure takes into account how much of the search space is visited. This is done by looking at the distance between a visited state and its nearest unvisited state. This is called the *gap*. If the maximum gap is large, this will be reflected in low coverage.

Fukunaga [21] found a correlation between the coverage and mobility of a search and the quality of the solution found. This illustrates that such properties may be instrumental in the selection or generation of operators for a search.

Smoothness

The last property we discuss is the *smoothness* of the search space for a given operator. A major difficulty faced by many search methods is getting stuck in local optima. SA addresses this problem by allowing occasional decreasing moves. However, for some local optima, escape may still be very unlikely, as several worse moves must be accepted first. Another tactic to solve this problem is to smooth the objective function directly, as is done in [23].



(a) Search space (b) Smoother search space

Figure 2.2: Visualisation of smoothness [23] for a minimization problem.

An important aspect to note here is that a local maximum for one operator may not be a local maximum for another operator. We expect that a good choice of operators will lead to a smoother search space with fewer local maxima where the search may get stuck. One possible way of defining such an operator is by viewing several operator moves as one single operator move, thus allowing for larger steps over the objective value space. Although this property is not outlined in detail, it is this intuition, that the concept of *chains* is built on.

2.3.2 Operators

Let us first define what an operator is exactly. As described in Section 2.2.2, in SA the solution space S for a problem P is traversed by moving from neighbour to neighbour. In practice, we do not have a predetermined set of neighbours for each state $s \in S$. Instead, we transform one state to a neighbouring state through the use of a function, an *operator.*

Definition 2. *Let $\mathcal{S}$ be the set of all solutions to a combinatorial optimization problem P and let L be the* ***location set*** *of σ. An* **operator** *σ is a function*

$$\sigma : \mathcal{S} \times L \to \mathcal{S}$$

that transforms one solution in the state space to another.

The exact value of the second variable, also referred to as the *location*, is usually drawn randomly from L according to some (often uniform) distribution. For example, for most operators for the Trapdoor problem, the location consists of which bit(s) are changed.

Another example of an operator is the Swap [13] operator as found in the Travelling Salesman problem (TSP). In TSP the solution consists of the order in which cities are visited along the route. The Swap operator selects two of these cities and swaps their place in the ordering. The location of the application of such an operator then consists of which two cities we choose to swap.

Each operator, with a corresponding location set, then defines a set of all solutions which a specific solution may be transformed into. We call these the *neighbours* of a solution.

Definition 3. *The* ***neighbourhood*** *of a state $s \in \mathcal{S}$ induced by operator $\sigma : S \times L \to S$ is defined as*

$$\mathcal{N}_\sigma(s) = \{\sigma(s,l) | l \in L\}.$$

For the Trapdoor problem, we will consider two operators: `bitflip` and `flip_to_1`. The `bitflip` operator selects a random bit and flips it. The `flip_to_1` operator selects a random bit and turns it to 1. For both of these operators the location set is $L = [n] \times [m]$.

Definition 4. *Let $X \in \chi^{n\times m} = \{0,1\}^{n\times m}$ and $i \in [n]$ and $j \in [m]$. Then, the operator*

$$\varphi(X, i, j) : \chi^{n\times m} \times [n] \times [m] \to \chi^{n\times m}$$

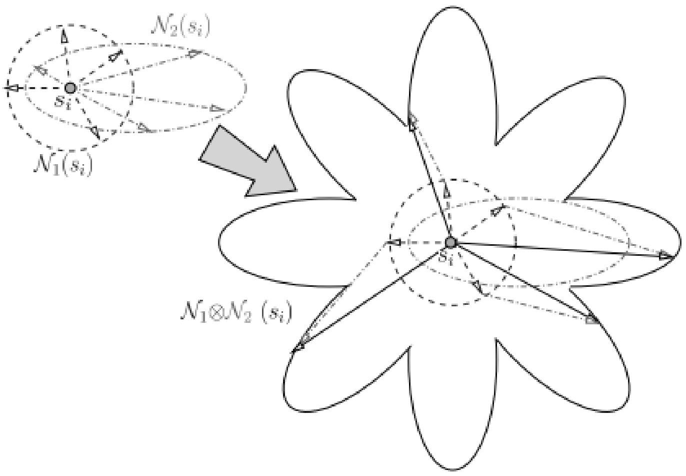


Figure 2.3: Effect on neighbourhood size of the composition of operators [16].

with

$$[\varphi(X,i,j)]_{uv} = \begin{cases} X_{uv} & \textit{if } (u,v) \neq (i,j) \\ (X_{uv}+1) \bmod 2 & \textit{otherwise,} \end{cases}$$

is called the ***`bitflip`*** *operator. When the context is clear, $\varphi(X,i,j)$ will usually be abbreviated to simply φ.*

Definition 5. *Let $X \in \chi^{n\times m} = \{0,1\}^{n\times m}$ and $i \in [n]$ and $j \in [m]$. Then, the operator*

$$\varphi_1(X,i,j) : \chi^{n\times m} \times [n] \times [m] \to \chi^{n\times m}$$

with

$$[\varphi_1(X,i,j)]_{uv} = \begin{cases} X_{uv} & \textit{if } (u,v) \neq (i,j) \\ 1 & \textit{otherwise,} \end{cases}$$

is called the ***`flip_to_1`*** *operator. When the context is clear, $\varphi_1(X,i,j)$ will usually be abbreviated to simply φ_1.*

Note that the φ_1 operator does not always change the solution. If the location of the application is a bit that is already 1 then the solution remains unchanged.

2.3.3 Relations

So far we have the tools necessary for a very simple concept of constructing new operators: successive application of multiple operators. This has the same effect as delaying the evaluation of the objective functions for multiple steps.

Definition 6. *Let $\sigma_1, \sigma_2 \in \Theta$ with corresponding location sets L_1, L_2. The* ***composition*** *of σ_1, σ_2 is defined as follows*

$$\begin{aligned} \sigma_2 \circ \sigma_1 : S \times L_c &\to S \\ (s,(l_1,l_2)) &\mapsto \sigma_2(\sigma_1(s,l_1),l_2) \end{aligned}$$

where $L_c = L_1 \times L_2$. In this case, σ_1 and σ_2 are referred to as the ***sub-operators*** *of $\sigma_2 \circ \sigma_1$.*

Figure 2.3 visualizes the effect such a composition of operators has on the size of the neighbourhood induced by this composition. Clearly, the neighbourhood corresponding to the composition is larger than the neighbourhoods corresponding to the two separate operators. This is a result of the following theorem.

Lemma 1. *Let $\sigma, \theta \in \Theta$ be two injective operators with corresponding locations sets L_1, L_2 respectively, such that $|L_1| \leq |L_2|$. Then*

$$|\mathcal{N}_\sigma(s)| \leq |\mathcal{N}_\theta(s)|\,.$$

Proof. The result follows directly from Definition 3. □

This means that for longer chains of operator compositions, the corresponding neighbourhood size will increase exponentially. This makes it harder to say something about the quality of an operator [16], as the induced neighbourhood may include many good options and many bad options. To counteract this, and make composite operators more specific, we introduce the concept of *relations*. In a composition of operators, relations help limit the number of neighbours, by limiting which locations later sub-operators may be applied to. To define this formally, we first need to define the concept of a *transition*.

As explained previously in Section 2.2.2.2, in SA we move from state to state by selecting an operator, generating its random component, applying the operator and potentially accepting the change. We will refer to these moves from state to state as a *transitions*.

Definition 7. *Let Θ be a collection of operators applied in annealing. Let $\mathcal{L}$ be the collection of corresponding location sets. Let $s \in \mathcal{S}$ be the collection of allowed states. We refer to the change from state s_i to state s_{i+1} during SA as a **transition**, written t_i. A transition is uniquely determined by*

- *its initial state, $s_i \in \mathcal{S}$,*
- *the operator that is applied, $\sigma \in \Theta$, and*
- *the location of application of the operator, $l \in L$, where $L \in \mathcal{L}$ is the location set corresponding to σ.*

We can thus write $t_i = (s_i, \sigma, l)$. We write $\mathcal{T}$ for the set of all possible transitions.

Between these transitions, we may now define relations.

Definition 8. *Let $\mathcal{T}$ be the set of possible transitions for a given heuristic. A **relation** is a function R*

$$R : \mathcal{T} \times \mathcal{T} \to \{\top, \bot\}.$$

When $R(t_i, t_j) = \top$, we say relation R holds between $t_i, t_j \in \mathcal{T}$, written $t_i \overset{R}{\sim} t_j$.

This concept of relations between transitions follows the same structure as the set-theoretical definition of a relation. For the relations used in this research, incidentally, all relations are reflexive, symmetrical, and transitive, meaning that they define equivalence classes. However, for general applications, this is not necessarily the case. For example, a relation based on geographical closeness is not necessarily transitive, or a relation based on a time interval taking place before another is not symmetric.

It is also important to note that relations do not necessarily only exist between sequential transitions. Figure 2.4 illustrates this.

In the HHs defined in Chapter **??**, we will use relations to limit the search for operators to sub-operators that are connected by some specific set of relations.

Definition 9. *Let $\mathcal{T}$ be the set of possible transitions for a given heuristic and $\mathcal{R}$ a set of relations. We say that two transitions t_i and t_j are **generally related** if there exists an $R \in \mathcal{R}$ such that $t_i \overset{R}{\sim} t_j$.*

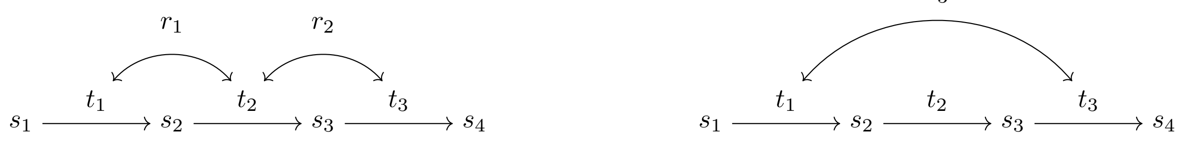


Figure 2.4: Visualisations of states s_i, transitions t_i, and relations r_i. Both cases are possible situations. Straight arrows signify transitions and rounded arrows signify relations.

Practically, in this research, we will only define relations based on the location. With slight misuse of notation we will thus usually speak of relations between locations, $l_i \overset{R}{\sim} l_j$.

Now, let us return to the Trapdoor example. As stated, the relations we define are only dependent on the location of the change (the bits that are potentially affected). Consider the Trapdoor problem of size $n \times m$ with $\Theta = \{\varphi, \varphi_1\}$. Then $\Theta = \mathcal{X}^{n\times m} \times \Theta \times [n] \times [m]$ is the set of all possible transitions. For this set of transitions, we can define the following simple relations.

Definition 10. *Consider a Trapdoor problem of size $n \times m$ with $\Theta = \{\varphi, \varphi_1\}$. Let $\mathcal{T} = \mathcal{X}^{n\times m} \times \Theta \times [n] \times [m]$. The **room relation**, $RM : \mathcal{T} \times \mathcal{T} \to \{\top, \bot\}$ is defined as follows*

$$RM(t_i, t_j) = \begin{cases} \top & \text{if } r_i = r_j \\ \bot & \text{otherwise,} \end{cases}$$

where $t_i = (s_i, \sigma_i, (r_i, c_i))$ and $t_2 = (s_j, \sigma_j, (r_j, c_j))$.

Definition 11. *Consider a Trapdoor problem of size $n \times m$ with $\Theta = \{\varphi, \varphi_1\}$. Let $\mathcal{T} = \mathcal{X}^{n\times m} \times \Theta \times [n] \times [m]$. The **column relation**, $CM : \mathcal{T} \times \mathcal{T} \to \{\top, \bot\}$ is defined as follows*

$$CM(t_i, t_j) = \begin{cases} \top & \text{if } c_i = c_j \\ \bot & \text{otherwise,} \end{cases}$$

where $t_i = (s_i, \sigma_i, (r_i, c_i))$ and $t_j = (s_j, \sigma_j, (r_j, c_j))$.

Simply put, these relations tell us if two operators were applied to a bit in either the same room or the same column, respectively[1]. We can also combine relations to create new relations. For example, combining the room and column relation gives us the bit relation.

Definition 12. *Consider a Trapdoor problem of size $n \times n$ with $\Theta = \{\varphi, \varphi_1\}$. Let $\mathcal{T} = \mathcal{X}^n \times \Theta \times [n]^2$. The **bit relation**, $BT : \mathcal{T} \times \mathcal{T} \to \{\top, \bot\}$ is defined as follows:*

$$BT(t_i, t_j) = CM(t_i, t_j) \wedge RM(t_i, t_j).$$

More generally, this structure of transitions and relations between transitions can be used to define a wide variety of relations in real-world problems, which we expect to be relevant. For example, in TSP we can use it to define a geographically close relation, or in the Vehicle Routing Problem, we can use it to define a same-tour relation. In many cases, identifying these potentially relevant relations may be much easier than defining new operators from scratch.

It is also worth noting that none of the relations used in this research are determined by the current state part of a transition. This allows us to regard transitions only as an operator and location tuple. However, the use of the current state in defining relations may be a relevant topic of future research, as it allows us to define a wider variety of relations.

2.3.4 Chains

Recall that we want to use relations to limit the potential neighbourhood size of composite operator functions. To this end, we define the following concept of a relation between operators. We will speak of a *chain* when referring to the sequential application of two operators such that a specific relation holds between their corresponding transitions. This is done by limiting the locations or states that each sub-operator is applied to. Let us make this more concrete for the case that relations are defined solely by locations.

Definition 13. *Let $\sigma_1, \sigma_2 \in \Theta$ with corresponding location sets L_1, L_2 and let $R \in \mathcal{R}$ be a relation defined on locations. The **chain** $\sigma_1 \xrightarrow{R} \sigma_2$ is a composite operator defined as follows:*

$$\begin{aligned} \sigma_1 \xrightarrow{R} \sigma_2 : S \times L(R, (\sigma_1, \sigma_2)) &\to S \\ (s, (l_1, l_2)) &\mapsto \sigma_2(\sigma_1(s, l_1), l_2), \end{aligned}$$

[1]Note that this does not necessarily mean that the bit is actually changed. For example, the φ_1 does not flip a bit that is already 1. To avoid this case, we could expand the definition to take into account the current value of the bit by looking at the initial state. This illustrates the use of including the current state in the transition.

where

$$L(R,(\sigma_1,\sigma_2)) = \{(l_1,l_2) \in L_1, L_2 : l_1 \overset{R}{\sim} l_2\}.$$

For the Trapdoor problem, an intuitive example of such an operator is the following:

$$\varphi_0 = \varphi_1 \xrightarrow{BT} \varphi.$$

When this chain is applied it randomly selects a bit and turns it to 1 by applying φ_1. It then flips the same bit (enforcing the BT relation) and turns it from 0 to 1. Such a composition thus corresponds directly to a $flip\_to\_0$ operator. We have

$$L(BT,(\varphi_1,\varphi_0)) = \{(l,l) | l \in L\}.$$

Such a structure need not only be applied to only two operations. We can generalize as follows

Definition 14. *Let $\{\sigma_i\}_{i\in[k]}$ be a set of operators from a set Θ, each with a corresponding location set L_i. Let $\{R_i\}_{i\in[k-1]}$ be a set of relations from a set $\mathcal{R}$. Define the following* ***chain***

$$\sigma = \sigma_1 \xrightarrow{R_1} \sigma_2 \xrightarrow{R_2} \cdots \xrightarrow{R_{k-1}} \sigma_k,$$

which is a composite operator defined as follows:

$$\sigma : S \times L\left(\{\sigma_i\}_{i\in[k]}, \{R_i\}_{i\in[k-1]}\right) \to S,$$

where

$$\sigma(s,(l_1,l_2,\cdots,l_k)) = \sigma_k(\sigma_{k-1}(\cdots\sigma_2(\sigma_1(s,l_1),l_2)\cdots,l_{k-1}),l_k),$$

and

$$L\left(\{\sigma_i\}_{i\in[k]}, \{R_i\}_{i\in[k-1]}\right) = \{(l_1,l_2,\cdots,l_k) \in L_1 \times L_2 \times \cdots \times L_k \mid l_1 \overset{R_1}{\sim} l_2, l_2 \overset{R_2}{\sim} l_3, \ldots, l_{k-1} \overset{R_{k-1}}{\sim} l_k\}$$

is the set of locations that enforce the relations. We will refer to the number of relations, $k-1$, as the ***length*** *of the chain σ.*

For the Trapdoor problem, an example of a chain of length 3 is the following:

$$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi,$$

which selects a room and 4 random bits in that room and flips them. Note that chains have a location set which is a subset of all possible locations, while compositions of operators have a location set which corresponds to all possible locations. According to Theorem 1 this means the neighbourhood induced by a chain is smaller than the neighbourhood by the simple composition of the same operators (if the operators are injective).

Theorem 2. *Let $\sigma_1,\sigma_2 \in \Theta$ be injective operators with corresponding locations sets L_1, L_2 and $R \in \mathcal{R}$. Let $\sigma_1 \xrightarrow{R} \sigma_2$ be a chain and let $\sigma_2 \circ \sigma_1$ be a composition of the operators. Then*

$$\left|\mathcal{N}_{\sigma_1 \xrightarrow{R} \sigma_2}(s)\right| \leq \left|\mathcal{N}_{\sigma_2\circ\sigma_1}(s)\right|.$$

Proof. The result follows directly from Lemma 1 and the fact that $|L(\sigma_1,\sigma_2,R)| \leq |L_1 \times L_2|$. $\square$

Using more or less restrictive relations thus allows us to define composite operators with smaller or larger induced neighbourhoods. It should be noted that in some cases the set $L\left(\{\sigma_i\}_{i\in[k]}, \{R_i\}_{i\in[k-1]}\right)$ could be empty. In this case, a chain would not be well-defined. However, in practice, we will only apply chains that have been observed in a set of transitions already and thus are by definition possible. On top of that, with the set of operators and relations for Trapdoor as defined in this chapter, all chains are well-defined.

To conclude, in this section, we have defined a structure which we will use to create composite operators. In the next section, several hyper-heuristic methods are discussed which generate such chains, built up from a set of operators, $\Theta = \{\varphi, \varphi_1\}$, and a set of relations, $\mathcal{R} = \{RM, RC, BT\}$. To avoid confusion between the set of operators used in SA (which may be chains as well) and the set of operators used as building blocks for these chains, we will commonly refer to the former as *global operators* and to the latter as *basis operators*. The set of basis operators is fixed as $\Theta = \{\varphi, \varphi_1\}$ throughout this thesis, while a different set of global operators may be used per SA run.

Chapter 3

Methods

This research aims to use the chain structure outlined in Section 2.3.4 to generate (or discover) effective operators for a given problem. One can imagine that in many cases, rather than designing a more complex operator from scratch, it is much easier to devise some basic operators and relations for a problem, which may or may not be useful for generating effective chain operators.

For longer chains, which are built from a variety of basic operators and relations, it is very likely that one of the chains included is an effective operator for the specific problem. However, the number of possible chains increases rapidly as we increase the number of basis operators and relations or allow longer chains. For n_r relations and n_o operators there are $n_o^{k+1} \cdot n_r^k$ chains of length k. In addition, as explained in Section 3.1, testing the performance of a single chain often requires several runs of Simulated Annealing (SA), and for larger problems, a single run of SA can take a very long time. The naive approach of chain-by-chain performance testing is therefore not practically realistic. Therefore, some method of determining which chains to test first is required.

Three distinct Hyper-Heuristics (HHs) aimed at solving this problem are presented in this chapter. All of the HHs return an order in which the chains are to be tested. One can imagine that, until the available time runs out, a developer would test the chains according to the returned order.

The chapter is structured as follows: First, we outline the method used to evaluate a chain's effectiveness, referred to as *performance*, in detail in Section 3.1.

In Section 3.2, we discuss the first HH, Priority Tree Search (PTS). PTS exploits the tree-like structure of chains with a Depth First Search inspired approach. When determining which node to expand next, it prioritises nodes associated with chains with higher performance. The algorithm is included as an example of a natural approach to discovering chains. However, as we will see in Chapter 5, the PTS method is not very effective at prioritizing high-performance chains.

The second approach, the Mining Hyper-Heuristic (M-HH), is discussed in Section 3.3. M-HH analyses the patterns in a single SA search to estimate which chains are likely to have higher performance. M-HH summarises a single search as a sequence of transitions. The algorithm then searches for chains that occur more frequently in search sections where some desired behaviour is observed. Based on the frequency with which a chain appears in such sections, a support value is calculated. These support values induce an order in which chains are tested. In Chapter 5 we will see that there is a positive relationship between the support and the performance of a chain. However, for many chains, a support value is never found.

Finally, in Section 3.4, we discuss the Population Mining Hyper-Heuristic (PM-HH). This approach aims to discover more chains by building on M-HH with a population system inspired by Genetic Programming algorithms. It does this by ensuring that promising chains occur more frequently in a SA search. The PM-HH selects the top chains discovered by the mining algorithm and includes them as global operators in a new search. It then mines this search sequence again for chains. PM-HH is discussed in more detail in Section 3.4. PM-HH indeed manages to discover more chains than M-HH, however, this comes at a cost to the relationship between support and performance, as we will see in Chapter 5.

3.1 Performance of a Chain

In order to test the performance of the different HHs developed in this chapter, we first need to define a way of assessing the quality of an operator, based on the SA Search it induces. This may be useful within an algorithm searching for operators, as we will see for PTS in Section 4, or to evaluate HHs and the operators they find, as we will do in Chapter 4. This section is dedicated to defining such a measure of *performance* of an operator (or chain).

There are two main aspects of a search that such a performance measure should at least take into account: (1) the time it takes to find a solution and (2) the objective value of the solution found. For example, a heuristic that takes two days to find a decent solution will likely be preferred to one that takes 10 years to find the best solution. The relative importance of time taken compared to the quality of the solution may differ per problem type or client. In some cases, solutions of a certain minimum quality must be found within a specific time, while in others the algorithm may run for weeks at a time to find the very best solution.

Since this research focuses on generating operators for an already existing implementation of SA, we can take both time and objective value into account by comparing to the time taken by the initial implementation and objective values of the solution it finds. By leaving one aspect fixed, we can consider how the other aspect is changed by the inclusion of an operator.

Let T_c, f_c be the average time taken and the average objective function of the current implementation of SA. If we fix T_c (for example, by fixing the total number of iterations used), then we can look at the average objective value returned when a new operator is included. When we fix the objective value f_c, we can look into how many iterations are needed until a solution of value f_c is found.

In this research, we chose to measure the performance of a chain by the average objective value found in the same number of iterations as the initial implementation. Testing the effect of alternative performance measures is left to further research.

3.1.1 Details of Implementation

The `Check_Perf` function is used to determine the performance of a chain c. The performance returned is the average objective value found for n runs of SA with two operators included: `bit_flip` and the chain c. Each operator is chosen with equal probability during the SA algorithm. The performance tells us something about how the objective value is affected by including the chain c along with the standard operators included. In this case, the standard operators included consist of the `bit_flip` operator. Algorithm 3 shows the pseudocode for the `Check_Perf` function.

Algorithm 3: Check Performance Function

```
Input  : Chain to test, $c$,
         Number of iterations of SA, $m$.
Output: Performance, $perf$.

Function Check_Perf($c$, $m$):
    $total\_obj \leftarrow 0$ ;                      ▷ Initialize total objective function
    for $i \leftarrow 1$ to $m$ do
        $sol \leftarrow$ SA(bit_flip, $c$) ;          ▷ Run Simulated Annealing
        $obj \leftarrow$ f($sol$) ;
        $total\_obj \leftarrow total\_obj + obj$ ;   ▷ Accumulate objective function
    $perf \leftarrow total\_obj/n$ ;                 ▷ Calculate average objective function
    return $perf$ ;
```

In the function, m is the number of times we run SA with the chain before we take the average objective value. Based on experiments found in Section 4.1, we set $m = 50$.

3.1.1.1 Implementation of Chains

When a chain is selected as an operator, all of the sub-operations that appear in this chain are applied in order. We enforce relations by limiting the location where operators are applied. For example, to enforce the room relation, the room number is kept constant compared to the previous sub-operator location, while a new column number is drawn randomly.

Counting Iterations

A relevant point of attention is how the number of iterations (at a given temperature) is counted when chains are included as global operators. Namely, a chain consists of several sub-operators. In our implementation each sub-operator which is performed counts as a single iteration. This means that a chain of length k requires $k + 1$ iterations. We elected to count iterations in this way to ensure that the exact same number of basis operators can be tried within SA, regardless of which global operators are included. This approach prevents an unfair artificial advantage from being given to chains over basis operators and ensures an even playing field when testing the performance of chain operators compared to basis operators.

Objective Value Calculation

One can imagine that using chains may speed up a search even when the same number of iterations is used because fewer objective value checks are required. However, as discussed in Section 2.2.2.5, it is often the case that the objective value is calculated incrementally by determining the difference in objective value rather than recalculating the total objective value for the new solution each time.

Our implementation of SA employs a similar approach. The difference in objective value is calculated by focusing only on the room(s) affected by the operator or chain application. Which rooms are affected can be easily determined by looking at the location of application. For chains, this means that the difference in objective value is calculated for each sub-operator in this manner and then summed. The move proposed by a chain operator is accepted based on this total difference in objective value.

Generating sequences

As will be discussed in more detail in Section 3.4, the PM-HH approach involves generating sequences derived from an SA search that incorporates chains as global operators. In this context, the sequence of transitions is recorded based solely on the application of basis operators, rather than the global operators (i.e., chains). For each sub-operator within an applied chain, we document the specific basis operator used, the location of its application (which may be constrained by defined relations), and the resulting effect on the objective function, which is conveniently calculated incrementally.

While this method focuses on logging only basis operator applications, an alternative approach could involve directly constructing sequences from chain operators. This may be an interesting topic for future research.

3.2 Priority Tree Search

The exploration of all chains of at most a given length is essentially a tree search problem. The Priority Tree Search (PTS) algorithm relies on this structure to discover effective chains. The algorithm is discussed in detail in this section.

A visualisation of an example of such a tree, associated with a set of basis operators and relations, can be found in Figure 3.1. In this tree, basis operators are shown as nodes and relations are shown as edges. Each path from the start node to a node in the tree corresponds to a different possible chain. The depth of a node then corresponds to the length of that chain. To thoroughly explore a tree up to a specified depth, classical algorithms such as Depth First Search [56, 26] can be utilized.

However, since our goal is to find chains with high performance in a limited time and tree size grows rapidly with chain parameters, it makes sense to limit the search to more promising parts of the tree. The `Priority_Tree_Search` function is specifically designed for this purpose, selectively expanding only the most promising chains (nodes) based on the highest performance observed thus far. The underlying assumption of this algorithm is that sub-chains of high-performance chains will also have a high performance.

Expanding a node means testing all possible expansions of the corresponding chain. Testing the performance of a chain requires several runs of SA with the chain included. As these runs are often very time-consuming, this aspect is likely to be the most time-consuming part of the algorithm. If there is limited

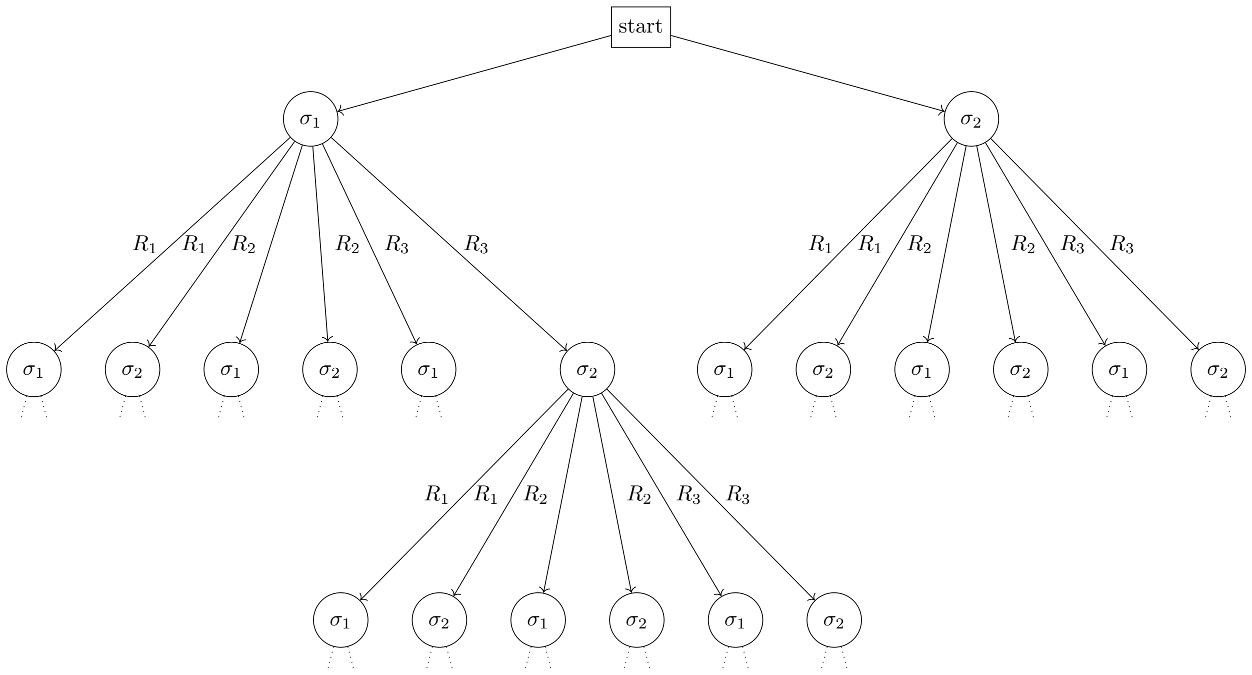


Figure 3.1: Tree diagram representing all possible chains for 2 operators (σ_1 and σ_2) and 3 relations (R_1, R_2 and R_3).

time available for testing chains we may want to limit the number of performance checks that are performed. To this end, we adapt the algorithm to only run for a maximum number of performance checks.

In Chapter 5 we find that while PTS does discover all chains up to a certain length, it does not effectively prioritize chains with high performance. We also find that the underlying assumption of the PTS algorithm is not generally true.

3.2.1 Description

The `Priority_Tree_Search` function starts by initializing a priority queue (implemented as a max-heap in Python) and a list to store good chains. Initial chain options (all basis operators) are pushed onto the heap with a high default performance value to ensure they are both expanded before the algorithm continues.

The algorithm enters a loop where it pops the chain with the highest performance from the heap. For this chain, the algorithm tries out all possible extensions by iterating through the basis operator and relation options. Each new chain formed is evaluated. The chain and its performance are then added to the list *found_chains*, and the count of tested chains is incremented.

If the number of tested chains reaches *max_to_test*, the function returns the list $found\_chains$. Otherwise, if the new chain's length is less than k, it is pushed onto the heap for further exploration.

The final list of $found\_chains$ is returned if the heap is empty or the maximum number of performance checks is achieved. The list is ordered in the same way as the chains were encountered in the algorithm. Details of the algorithm can be found in Algorithm 4.

3.2.2 Parameters

The algorithm takes the following parameters as input:

- **Set of basis operators** (Θ): This is the set of all possible operators that can be applied to extend a chain. Each operator represents a potential action or transformation that can be applied to the current solution in SA.

Algorithm 4: Priority Tree Search

```
Input  : Set of operators, $\Theta$,
         Set of relations, $\mathcal{R}$,
         Maximum chain length, $k$,
         Maximum number of performance checks, $max\_to\_test$.
Output: List of chains with corresponding performance, $found\_chains$.

Function Priority_Tree_Search($\Theta$, $\mathcal{R}$, $k$, $max\_to\_test$):
    $found\_chains \leftarrow list()$ ;                                      ▷ Initialize
    $heap \leftarrow heap()$
    for $\theta \in \Theta$ do
        $heap$.push($\infty$, $\theta$);
    $i \leftarrow 0$
    while $max\_heap$ do
        $\theta, perf \leftarrow max\_heap$.pop() ;                ▷ Pop best chain found so far
        for $(R, \sigma) \in \mathcal{R} \times \Theta$ do
            $new\_chain \leftarrow (\theta \xrightarrow{R} \sigma)$
            $perf \leftarrow$ Check_Perf($new\_chain$) ;                  ▷ Evaluate new chain
            $found\_chains$.append($new\_chain, perf$) ;    ▷ Append new chain and performance
            $i \leftarrow i + 1$
            if $i \geq max\_to\_test$ then
                return $found\_chains$
            if $|new\_chain| \leq k$ then
                $heap$.push($perf, new\_chain$) ;                          ▷ Push chain to heap
    return $found\_chains$
```

- **Set of relations** ($\mathcal{R}$): This is the set of all possible relations that can exist between operators in the chains. Relations define how operators can be combined or sequenced together.
- **Maximum chain length**, (k): The maximum length that a chain is allowed to reach.
- **Maximum number of chain checks**, (max_to_test): The maximum number of chains that the algorithm is allowed to test (excluding the initial operators).

In this research, we will use the basis operators $\Theta = \{\varphi, \varphi_1\}$ as described in Section 2.3 and relations $\mathcal{R} = \{RM, CM, BT\}$ as described in Section 2.3.3. We will set $k = 4$, and $max\_to\_test = 3108$, so that all chains of length 4 will be found. In this way, we can determine the number of performance checks necessary before each chain is found. PTS is tested in detail in Section 4.2.

3.3 Mining Hyper-Heuristic

As we will see in Chapter 5, PTS is not very effective at discovering chains with high performance. For this reason, an alternative HH was developed: the Mining Hyper-Heuristic (M-HH), which uses more of the available information contained in a SA run to determine which chains to test first.

In PTS, the only information from a run of SA used is the objective value of the solution returned. However, a lot more information is contained in a single run of annealing. With the right choice of global operators, it is likely that many chains appear naturally in a single SA search. In addition, since a single run of SA often takes a long time, it makes sense to exploit all information we can from it, so that fewer SA runs may be necessary.

As discussed in Section 2.3.1, the effect of an operator (or chain) on the objective value during search may be relevant for predicting the performance of the chain. The M-HH aims to discover chains which are expected to smooth the objective value space, as they can directly cause large spikes in objective value, instead of relying on first accepting a series of worse moves with small probability.

These support values are determined by analyzing short sequences of transitions within a single SA run where desirable behaviour is observed. In this implementation, the desired behaviour is a large increase in the objective value. The algorithm aims to discover which chains are likely to have caused this spike in objective value by determining which chains appeared right before it and thus could have caused it. The algorithm rests on the assumption that chains that appear more often right before large spikes are more likely to have caused the spikes. This likelihood is summarized in the *support* value, as defined in Definition 18. A chain with a higher support value appears more often right before large spikes than it does normally in search.

The algorithm consists of 4 main phases:

1. Generate sequences: A *sequence* contains all information we use from a single SA run. It consists of an ordered list of accepted transitions from an SA run and their corresponding changes in objective value.

2. Find sub-sequences: The sequences are cut into smaller sections of fixed length, referred to as sub-sequences. These sub-sequences are placed into two groups: (1) *good sub-sequences*, which are sections with desirable behaviours, and (2) *normal sub-sequences*, which are generic sections.

3. Find frequency of chains for all sub-sequences: In this step, we count the chains as they appear in a sub-sequence. We only count chains whose final sub-operator is the final operator of the sub-sequence, the *lynchpin*.

4. Calculate support for chains: The *support* value is a value which captures how much more likely a chain is to appear in a good sub-sequence than in a normal sub-sequence.

Each phase and its related concepts are discussed in more detail in the Sections 3.3.1 to 3.3.4 below. In Section 3.3.5 all parameters used in the HH are summarized. In figure 3.3, a simplified version of the algorithm is shown in diagram form.

In Chapter 5 M-HH is tested in detail. We find that a significant positive relationship exists between the support and performance of the discovered chain, which suggests that such a mining-based technique is a promising approach. However, many chains were never discovered by the algorithm.

3.3.1 Generating Sequences

The first step of the algorithm is generating a number of *sequences*. A sequence consists of a set of transitions as they occur in a run of SA, and their corresponding changes in objective value. We can thus generate a sequence by running SA with a certain set of operators and keeping track of accepted changes and the corresponding changes in objective value. A sequence can then be summarized as an ordered list of tuples (T, Δ), where $T \in \mathcal{T}$ and Δ is the change in objective value brought about by this transition. The length of the sequence is equal to the number of accepted moves for that iteration of SA and may vary.

1	-1	1	-1	1	-1	9	1	-1	-1	1	-1	1	1	1	-1	-1	9
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}

Table 3.1: Example of a sequence of length 18.

3.3.2 Sub-sequences

These generated sequences are then cut into smaller sections of a fixed length, *sub-sequences*.

Definition 15. *A **sub-sequence** is a contiguous portion of a larger sequence that is cut to a fixed length, specified by len_sub.*

We create two sets of sub-sequences: good sub-sequences and normal sub-sequences. The set of good sub-sequences are sub-sequences where the desired behaviour in terms of objective value is observed at the final transition (also referred to as the *lynchpin*).

Definition 16. *The **lynch-pin** is the last transition that occurs in a sub-sequence.*

Such desired behaviours may vary, depending on what kind of behaviours the designer is interested in. Examples include a large spike in the objective value, a suggested change being accepted or an improvement after a long time without improvements. In this research we will employ the following definition of a good sub-sequence:

Definition 17. *A* ***good sub-sequence*** *is a sub-sequence for which the difference in objective value at the lynch-pin is equal to the largest possible improvement in objective value occurs, namely* $\Delta = (C + B - 1)$.

Recall that we work with an instance of the Trapdoor Problem with number of rooms equal to $R = 6$, number of columns equal to $C = 6$ and bonus $B = 4$. This means that a good-subsequence is any sub-sequence which ends with an increase in objective value of size $\Delta = (C + B - 1) = 9$. Table 3.2 shows an example of two good sub-sequences of length 6.

This definition was chosen as a simple starting point for testing the potential use of mining-based HHs such as M-HH. It captures the relatively rare behaviour of escaping a local optimum which we want to learn from, while still remaining easy to deduce from a given sequence. While results discussed in Section 5.3 show that the definition is effective for M-HH, in later experiments discussed in Section 5.4, we find that a more complex definition may be useful to avoid discovering back-and-forth chains which do not change the solution. This would be an interesting topic for future research.

1	-1	1	-1	1	-1	9	1	-1	-1	1	-1	1	1	1	-1	-1	9
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}

Table 3.2: Example of a sequence of length 18 with good subsequences of length 6 shown outlined. This sequence also contains 12 normal sub-sequences of length 6.

We also define a set of *normal sub-sequences* to use as a baseline to compare to. These are all sub-sequences of length *len_sub*. These can be found with a simple sliding window technique [12]. Note that the set of normal sub-sequences also contains the set of good sub-sequences.

The division into two sets makes it possible to compare which operators seem to occur more in sections with desirable behaviour. Here, the underlying assumption is that the chains which occur more in these good-sub-sequences than in normal sub-sequences are the cause of the desirable behaviours and are more likely to improve overall performance.

3.3.3 Finding Chains

The next step is to find the chains that appear in a sub-sequence. As stated, chains must end with the *lynchpin* operator. This is motivated by that fact that the lynch pin operator is the only operator of which we are sure it had something to do with the change in objective value observed at the corresponding transition. If we assume such a change in objective value is caused by some chain, then that chain must end with the lynchpin operator.

It is important to realize that when several operators cause a spike in objective value, they occur *successively* but not necessarily *consecutively.* That is to say, the transitions or moves (operators) that lead to a significant improvement in the objective value may happen in a specific order (successively) during the optimization process, but they do not have to occur one right after the other without any interruptions (not consecutively). For example, consider a chain that flips several bits to zero in the same room. If a bit is flipped in a different room in between its sub-operators, we expect this to have no effect on whether the chain incurs a large change in objective value or not. This means the algorithm must discover all chains that appear in the sub-sequences, where sub-opteration of the chain need not follow directly one after the other. We have taken this into consideration by allowing relations to exist between non-consecutive transitions as well.

By working back from the lynchpin we can design a recursive algorithm which finds all chains 'contained' in the sub-sequence, i.e. all chains which appear in the sub-sequence and end with the lynchpin operator. The details of the algorithm can be found in Algorithm 5.

The algorithm begins by identifying the lynchpin operator, which is the last operator in the sub-sequence (sub). We also initialize an empty list ($found\_chains$) to store the chains identified during the search.

Algorithm 5: Find Chains

```
Input  : A sub-sequence, $sub$,
         Length of sub-sequences, $len\_sub$,
         Maximum length of chain, $len\_chain$,
         Level of recursion, $cur\_len$.
Output: A list of chains that appear in the sub-sequence, $found\_chains$.
Function Chain_Search($sub$, $len\_chain$, $cur\_len=0$):
    $lynch\_pin \leftarrow sub[-1]$ ;                                  ▷ Initialize
    $\sigma_l \leftarrow lynch\_pin[1]$ ;
    $found\_chains \leftarrow list()$ ;
    if $cur\_len == len\_chain$ or $len(sub) == 1$ then
        return $found\_chains$ ;                                       ▷ Base case
    for $i \leftarrow len(sub) - 1$ to 0 do
        $step \leftarrow sub[i]$ ;                   ▷ Move backwards from the lynchpin
        $\sigma_s \leftarrow step[1]$;
        for $R \in \mathcal{R}$ do
            if $R(step, lynchpin) == \top$ then
                $chains.append(\sigma_s \xrightarrow{R} \sigma_l)$ ;
                $sub\_chains \leftarrow$ Chain_Search($sub[: i+1], len\_chain, cur\_len+1$) ;   ▷ Recurse
                for $sub\_chain \in sub\_chains$ do
                    $found\_chains.append(sub\_chain \xrightarrow{R} \sigma_l)$ ;   ▷ Combine chains
    return $found\_chains$ ;
```

If the current recursion level (cur_len) equals the maximum length of the chain (len_chain), or if the length of the sub-sequence is 1, the algorithm returns the $found\_chains$ list. This serves as the base case for the recursion.

The algorithm iterates backward through the sub-sequence starting from the lynchpin transition. For each transition in the sub-sequence, it evaluates if the transition is generally related to the lynchpin transition. If this is the case the algorithm recursively searches for additional chains in the truncated sub-sequence (start to current transition). Any new chains found in this recursive step are combined with the current relation and lynchpin operator and added to the $found\_chains$ list.

After completing the recursive search, the algorithm returns the list of all identified chains ($found\_chains$) that end with the lynchpin operator. Some chains may appear more than once in this list.

For further clarification, a single recursion of the algorithm is shown in Figure 3.2. Here, transitions are shown as nodes labelled with their corresponding operators. Relations between transitions are shown as edges for two different relations, shown in red and blue. The lynchpin for each recursion is shown as the green node.

3.3.4 Calculating support

Finally, the *support* value for all found chains is calculated according to Definition 18. The support value reflects how much more likely it is that a chain is found in a good sub-sequence as opposed to a normal sub-sequence. Such a support value can be used as an indication of the potential of a chain.

Our definition of support is inspired by the concept of support used in frequent itemset mining and association rule learning [1]. In these fields, the goal is to identify sets of items that frequently co-occur within a given dataset. The primary objective is to find all itemsets that meet a specified minimum support threshold. Here, the support value of an itemset is defined as the proportion of transactions in the dataset that contain that itemset.

To apply this to chains within sub-sequences, we can think of chains as itemsets, good sub-sequences as transactions, and all sub-sequences as the dataset.

However, we've made several adaptations to make the definition of support more appropriate for our context. Firstly, while an itemset may appear only once in each transaction, a chain can appear multiple times within a single sub-sequence. Therefore, we calculate the average frequency of a chain rather than simply counting the sub-sequences in which it appears. Secondly, to account for chains that naturally occur more frequently, the support of a chain is normalized by dividing by its average frequency in normal sequences. This leads to the following definition:

Definition 18. *Let $\mathcal{G}$ be the set of good sub-sequences and $\mathcal{N}$ the set of normal sub-sequences. The **support** of a chain c is equal to*

$$\mathrm{Sup}(c) = \frac{\sum_{G\in\mathcal{G}} \mathrm{freq}_G(c)/|\mathcal{G}|}{\sum_{N\in\mathcal{N}} \mathrm{freq}_N(c)/|\mathcal{N}|} = \frac{|\mathcal{N}| \sum_{G\in\mathcal{G}} \mathrm{freq}_G(c)}{|\mathcal{G}| \sum_{N\in\mathcal{N}} \mathrm{freq}_N(c)}.$$

Here, $freq_H(c)$ is the frequency of chain c in sub-sequence H.

The frequency of a chain in a sub-sequence corresponds to how often the chain appears in the returned list from `Chain_Search`. In our implementation, the support is only calculated for chains with a frequency higher than $min\_freq = 20$ in both good and normal sub-sequences to avoid large outliers.

3.3.5 Parameters

To implement the algorithm, the following parameters need to be configured:

- **Number of sequences** (*num_seq*): The number of SA runs we use to generate sequences. This is equal to the number of sequences generated. By setting this value higher, the resulting support values become less variable. However, this comes at the cost of a longer running time.
- **Sub-sequence length** (*len_sub*): The length of sub-sequences. A higher sub-sequence length means we can look further back from the lynchpin. However longer sub-sequences also increase the complexity of the `Chain_Search` algorithm.
- **Chain length** (*len_chain*): The maximum length of chains considered in the mining process. This parameter restricts the depth of the recursive chain search algorithm so that we only find chains with maximum length equal to *len_chain*.
- **Minimum frequency** (*min_freq*): The threshold for the minimum number of times a chain must appear in both good and normal sub-sequences to be considered in the support calculation.
- **Basis operators** (Θ): The set of global operators which we use in SA. These are also the only operators which will appear in the found chains.
- **Relations** ($\mathcal{R}$): The set of possible relations that define how transitions are connected to form chains. These relations should be relevant to the problem domain.

In the implementation of the algorithm, we have set $num\_seq = 500$. This value was chosen to illustrate what the M-HH can do in roughly the same time as it would take to check as few as 10 chains. During experimentation, we tested the technique for various values of *len_sub* and *len_chain* to determine their effect. As stated in Section 3.3, we set $min\_freq = 20$ to avoid large outliers. All chains are built up from a set of basis operators, $\Theta = \{\varphi, \varphi_1\}$, and a set of relations, $\mathcal{R} = \{RM, RC, BT\}$.

3.4 Population Mining Hyper-Heuristic

As will be discussed in detail in Section 5, the mining technique exhibits promising results, particularly in terms of the relationship between support and performance of a chain, and the quality of the ordering induced by support value.

However, a significant issue remains: for a large number of chains the support value is never calculated. This is a result of the chains not appearing often enough in good sub-sequences, i.e. less than $min\_freq = 20$. Initial experimentation showed that even for lower values of min_freq the problem persisted. By modelling the solution space as a Markov chain, we can easily determine that the likelihood of a longer chain appearing in a sub-sequence is very small, even for higher temperatures at the start of the search. For longer chains, this issue is exacerbated.

To address this issue, we introduce the Population Mining Hyper-Heuristic (PM-HH), an extension of the original M-HH. This approach combines mining with a mechanism that ensures promising chains appear more frequently, thus providing more information to build upon. This mechanism employs a population system where chains with the highest support are directly used as operators in SA to generate new sequences. These generated sequences are then mined again for chains with high support values, which can be used as operators in subsequent iterations.

In Chapter 5 we find that the PM-HH effectively discovers more chains than M-HH. However, this improvement comes with a limitation: we find that the criteria for defining good sub-sequences is relatively simplistic, leading to many ineffective chains being assigned a high support value.

3.4.1 Description

Algorithm 6 presents the pseudocode for PM-HH. Additionally, Figure 3.3 illustrates the mining mechanism with and without the population system, providing a comparison of the two approaches.

The algorithm starts by generating several sequences using SA with a set of basis operators, just like in M-HH. These sequences are then mined to identify chains and calculate their support values, again similar to M-HH. However, instead of simply returning all the chains and their support values, we select a subset of chains with the highest support values. The number of chains selected is determined by the *pop_size* variable.

Next, we generate a new set of sequences using both the selected high-quality chains and the original basis operators as global operators in SA. These new sequences are mined again to find chains and calculate their support values. This process of generating sequences, mining for chains, and calculating support values is repeated multiple times, as specified by the *num_gen* variable.

After completing the specified number of generations, the algorithm returns the final list of chains and their support values from the last generation.

Algorithm 6: Population Mining Hyper-Heuristic

```
Input  : Number of generations, $num\_gen$,
         Population size, $pop\_size$.
Output: A list with $(chain, support)$ tuples, $quality\_chains$

Function Pop_Chain_Search($\Theta_b$, $num\_gen$, $pop\_size$, ...):
    $\Theta \leftarrow \Theta_b$ ;                                          ▷ Initialize population
    $n_o \leftarrow len(\Theta_b)$ ;
    $P \leftarrow [1/n_0] * n_0$ ;
    for $i \leftarrow 1$ to $num\_gen$ do
        $seqs \leftarrow$ Generate_Seqs($num\_seq, \Theta, P$) ;               ▷ Generate sequences
        $quality\_chains \leftarrow$ Mine_Chains($len\_chain, \dots$) ;      ▷ Find chains and support values
        $\Theta, P \leftarrow$ Set_Pop($pop\_size, quality\_chains, \Theta_b$) ;   ▷ Update population
    return $quality\_chains$ ;
```

Here, the `Generate_Seqs` function generates *num_seqs* sequences by running SA with operators Θ. Each operator is chosen with a probability determined by P. The `Mine_Chains` function returns a list of chains with an associated support value by following the steps outlined in Section 3.3. The `Set_Pop` function determines the population for the next generation, as outlined in the next section.

3.4.2 Population Selection

The population consists of a list of operators and a set of probabilities corresponding to the likelihood that each operator is chosen. In our population, we ensure that half of the time a basis operator is chosen and half of the time one of the top *pop_size* mined chains is chosen. Each of the basis operators is chosen with equal probability and each of the chains is chosen with equal probability.

This means that for a set of n basis operators $\Theta_b = (\sigma_1, \ldots, \sigma_n)$ and a set of m chains $\Theta_c = (\theta_1, \ldots, \theta_m)$ to be placed in the population, we get operator vector:

$$\Theta_p = (\sigma_1, \sigma_2, \ldots, \sigma_n, \theta_1, \theta_2, \ldots, \theta_m)$$

and probability vector:

$$P = \left(\underbrace{\frac{1}{2n}, \frac{1}{2n}, \ldots, \frac{1}{2n}}_{n \text{ times}}, \underbrace{\frac{1}{2m}, \frac{1}{2m}, \ldots, \frac{1}{2m}}_{m \text{ times}} \right)$$

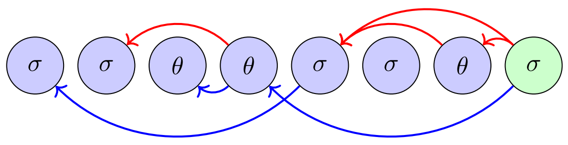
as population.

3.4.3 Parameters

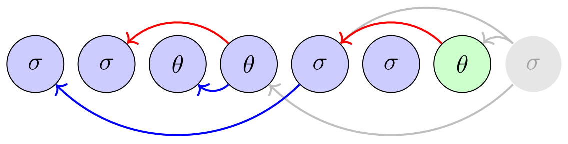
Since the population mechanism repeatedly performs the mining process it also uses all mining parameters (see Section 3.3.5). On top of that, two new parameters are introduced:

- **Population size** (*pop_size*): The number of chains with the highest support which we use in the next iteration of generating sequences.
- **Number of generations** (*num_gen*): The number of population updates we perform.

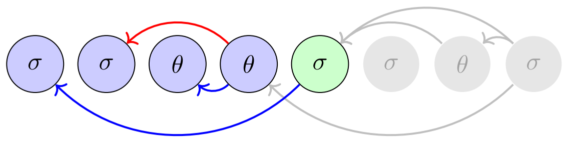
In this research, we will only test the performance of PM-HH for *num_gen* = 2, *pop_size* = 10, *len_sub* = 15, *len_chain* = 4 and *num_seq* = 250. All other parameters are the same as for the M-HH algorithm so that the effectiveness of the algorithms can more easily be compared. The experiments performed with these parameters are not meant to be a complete exploration of the population mining technique and all possible parameter settings. We only aim to show that the approach is promising in terms of increasing the number of discovered chains.



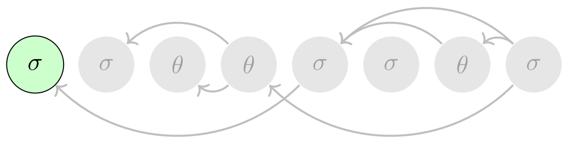
(a) Start of algorithm. The set $found\_chains$ is empty. The algorithm starts by moving back through the sub-sequence and checking relations.



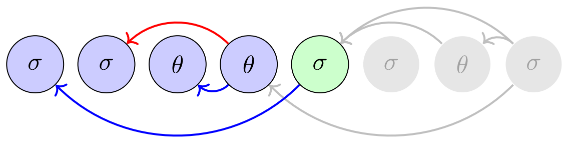
(b) A first R relation (red) is discovered and the algorithm is applied to a shorter sub-sequence. Within this recursion, the set $found\_chains$ is initialized as empty.



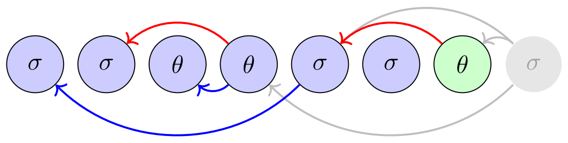
(c) A first B relation (blue) is discovered and the algorithm is applied again to a shorter sub-sequence. Within this recursion, the set $found\_chains$ is initialized as empty.



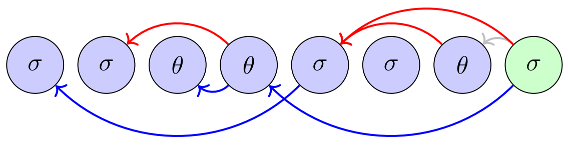
(d) Sub-sequence length is equal to 1 (base case), so the algorithm returns $\{\}$. This initiates a backwards pass across the recursions.



(e) The set $found\_chains$ is updated to $\{\sigma \xrightarrow{B} \sigma\}$. No more relations are found within the sub-sequences. The algorithm returns $\{\sigma \xrightarrow{B} \sigma\}$.



(f) The set $found\_chains$ is updated to $\{\sigma \xrightarrow{B} \sigma \xrightarrow{R} \theta, \sigma \xrightarrow{R} \theta\}$. No more relations are found within the sub-sequence. The algorithm returns $\{\sigma \xrightarrow{B} \sigma \xrightarrow{R} \theta, \sigma \xrightarrow{R} \theta\}$.



(g) The set $found\_chains$ is updated to include $\{\sigma \xrightarrow{B} \sigma \xrightarrow{R} \theta \xrightarrow{R} \sigma, \sigma \xrightarrow{R} \theta \xrightarrow{R} \sigma, \theta \xrightarrow{R} \sigma\}$. The algorithm will now continue checking the sub-sequence for relations.

Figure 3.2: Visualisation of a single recursion of chain search algorithm for 2 operators, θ and σ, and 2 relations, R and B (shown in red and blue arrows respectively). The lynchpin operator is shown in green.

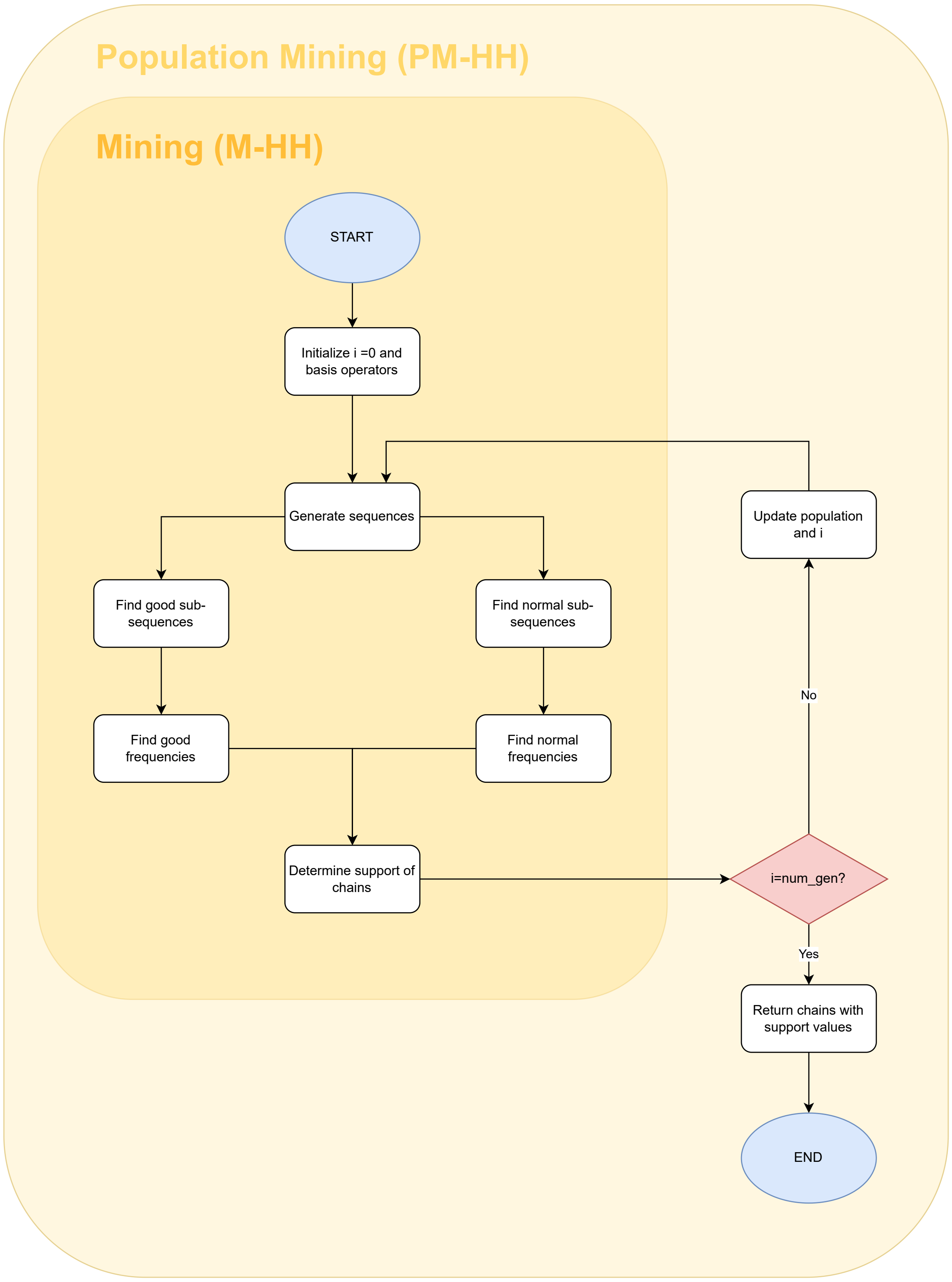

Population Mining (PM-HH)
Mining (M-HH)
START
Initialize i =0 and basis operators
Generate sequences
Find good sub-sequences
Find normal sub-sequences
Find good frequencies
Find normal frequencies
Determine support of chains
i=num_gen?
No
Update population and i
Yes
Return chains with support values
END

Figure 3.3: Simplified diagram of steps in M-HH and PM-HH.

Chapter 4

Preliminary Experiments

Before we can perform the experiments used to test the effectiveness of the developed Hyper-Heuristics (HHs) it is first necessary to address some preliminaries, which are discussed in this chapter.

In Section 4.1, we discuss a small-scale preliminary experiment. Based on the result of this experiment we fix certain parameters: the number of runs used to test the performance of a chain, m, and the basis performance, f_b. These values are used in later experiments to evaluate the effectiveness of different HHs.

In Section 4.2, we test the performance of all chains of chain length at most 4, for a set of basis operators and relations. We do this for two reasons. First, to save time in later experiments and second, to gain insight into the usefulness of the chain structure. We find that many chains significantly improve the performance when compared to the case without chains. In addition to this, we find that several chains exist which cause the optimal solution to always be found over 50 runs of Simulated Annealing (SA).

4.1 Determining Parameters

Before commencing our larger experiments, it is essential to determine the optimal number of runs needed to accurately evaluate a chain's performance. This section presents a small-scale preliminary experiment designed to inform our choice.

Based on the outcomes of this experiment, we have set the number of runs to $m = 50$, as a standard for subsequent performance testing. Furthermore, we establish a baseline performance, $f_b = 51.71$, which serves as the threshold above which chains are considered to be significantly effective in enhancing performance. This baseline will play a critical role in identifying and evaluating the effectiveness of different chains in the experiments that follow, as is discussed in more detail in Section 5.1.

4.1.1 Preliminary Experiment

To gain more insight into the variability of the objective value and the time required for SA runs, we conducted 100 runs of SA with chains of varying lengths. These chains are constructed by using only the bitflip operator and the column relation, as these are not expected to significantly affect the performance of SA. The set of global operators used in SA consists of the bitflip operator and such a chain, each of which is selected with probability 1/2, similarly to how the performance of chain is defined.

The results are presented in Tables 4.1 and 4.2.

4.1.1.1 Discussion

Several points are noteworthy in these results. First, the average objective value of a solution found by SA with only the bitflip operator is 50.02 (±4.93). As discussed in Section 2.1, a room consisting of only 0's is worth 10 in terms of objective value and a room consisting of only 1's is worth 6 in terms of objective value. An average objective value of 50, then means that on average 3.5 of the 6 rooms contain only 0's.

Second, we note that the average solution quality ($\overline{f}$) decreases when longer chains are included as global operators. When a chain of length 4 is included the average solution quality is 44.93 (± 3.75), while in the case with only the `bitflip` operator, the average solution quality is 50.02 (± 4.93). The non-overlapping

Chain	$\overline{f}_i$	Std Dev	95% CI
φ	50.02	4.93	(49.05, 50.99)
$\varphi \xrightarrow{CM} \varphi$	48.31	3.80	(47.56, 49.06)
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	46.78	4.09	(45.98, 47.58)
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	45.94	4.07	(45.14, 46.74)
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	44.93	3.75	(44.19, 45.67)

Table 4.1: Summary of mean objective values of the solution found over 100 runs of SA with `bitflip` and various chains included as global operators.

Chain	$\overline{T}_i$	Std Dev	95% CI
φ	0.78	0.04	(0.78, 0.79)
$\varphi \xrightarrow{CM} \varphi$	0.56	0.04	(0.55, 0.57)
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	0.45	0.02	(0.45, 0.45)
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	0.38	0.02	(0.38, 0.39)
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	0.34	0.02	(0.34, 0.34)

Table 4.2: Summary of mean running time (in seconds) over 100 runs of SA with `bitflip` and various chains included as global operators.

confidence intervals suggest that these changes are significant, which contradicts our initial assumption that such chains would have no effect on the objective value of the solution discovered. Such a decrease in objective value is likely a consequence of the column relation limiting the exploration of the solution space, by ensuring only bits in one column are changed.

Third, for longer chain lengths included as global operators, the average time required to run SA appears to decrease, from 0.78 (± 0.04) for a chain of length 0 to 0.34 (± 0.02) for a chain of length 4. The non-overlapping confidence intervals indicate that these changes are significant. As discussed in Section 3.1 the number of sub-operators allowed (or total iterations) is kept the same in the case with and without chains, so it is not likely that this is the cause of the decrease in time spent. As discussed in Section 3.1 the objective value is also calculated incrementally per sub-operator, and thus not likely to be the cause either.

It is possible that the selection process of which operator to use is the cause of the difference in time needed. For long chains, the choice of the next few operators to use has been predetermined, meaning that several steps of randomly drawing a number can be skipped. However, we expect that the time needed to draw a random number in Python is negligible compared to the other steps performed in the algorithm.

One of the most time consuming steps in the algorithm consists of copying the best solution found when a new better solution is discovered. In this step, all 36 bits must be copied into a new instance of the Trapdoor problem. We expect that in the early stages of the algorithm when we often discover new better solutions, this step occurs less in a SA which includes longer chains. Namely, the best solution is only updated once the whole chain has been applied, and not after every sub-operator which may each individually also lead to a better solution than encountered so far. Based on this explanation we will assume that the time needed to test a chain of fixed length is constant, and account for this when estimating the total time needed for testing all chains in the next section.

It is worth noting that the observed difference in running time of SA for different global operators used is a result of the way the SA algorithm was implemented, and may not generalize to alternative implementations of SA.

4.1.2 Setting m

Next, we must determine m, which corresponds to the number of SA runs the performance value of a chain is averaged over, as discussed in Section 3.1. Increasing the number of runs reduces the variability in performance measurements, enhancing their reliability. However, more runs also require more time.

Based on the previous experiment, we can estimate the time needed to test the performance of all chains up to length 4, by running SA m times:

$$m \cdot \sum_{i=0}^{4} n_o^{i+1} \cdot n_r^i \cdot \overline{T}_i = 1087.49 \cdot m \text{ seconds.}$$

Here $\overline{T}_i$ is the estimated time necessary to run SA for a chain of length i, which can be found in Table 4.2, $n_r = 3$ is the number of relations and $n_o = 2$ is the number of operators. To ensure that determining the performance values of these chains does not take longer than a day we fix $m = 50$. For this value of m, testing the performance of all chains of at most length 4 would take roughly 15 hours.

4.1.2.1 Size of Confidence Intervals

To gain some understanding into the significance of the performance values measured for different chains, we estimate the size of the 95% confidence interval, based on the results from the preliminary experiment. Details on the statistical methods used can be found in [15].

In calculating this confidence interval we assume that the data is distributed normally. However, since the objective value of a solution necessarily lies between 6 and 60, this may not necessarily be the case. For this reason, the values calculated below should be seen more as an indication of significance, rather than a detailed proof.

The confidence interval of some measured performance $\overline{f}$ is equal to

$$\left[\overline{f} - ME, \overline{f} - ME\right]$$

where ME is the margin of error for a 95% confidence interval, defined as follows:

$$ME = t_{\alpha/2,m-1} \cdot \frac{SD}{\sqrt{m}}$$

where $m = 50$ and SD is the standard deviation of the objective value over the 50 SA runs. For a 95% confidence interval and $m - 1 = 49$ degrees of freedom, the critical value $t_{\alpha/2,m-1}$ can be found using the t-distribution table. For a 95% confidence level, $\alpha = 0.05$, and $\alpha/2 = 0.025$, the critical value for $t_{0.025,49} \approx 2.01$.

We can estimate the maximum value of the margin of error by using the maximum standard deviation measured in the above experiment, $s_{\max} = 4.93$. The margin of error is then at most:

$$ME = t_{\alpha/2,m-1} \cdot \frac{SD}{\sqrt{m}} \leq t_{\alpha/2,m-1} \cdot \frac{s_{\max}}{\sqrt{m}} = 2.01 \cdot \frac{4.93}{\sqrt{50}} = 1.4.$$

This means that with 95% certainty the measured performance differs at most 1.4 from the actual performance of that chain. Since the actual performance lies in the interval $[6, 60]$, we deem a maximal difference of 1.4 to be sufficient.

4.1.2.2 Average Chain Check Time

Based on the previous experiment and the value of m we can estimate the time need to check the performance of a random chain (of at most length 4) by dividing the estimated time spent on all chains, by the number of chains, as follows:

$$m \cdot \frac{\sum_{i=0}^{4} n_o^{i+1} \cdot n_r^i \cdot \overline{T}_i}{\sum_{i=0}^{4} n_o^{i+1} \cdot n_r^i} = 17 \text{ seconds.}$$

Here $\overline{T}_i$ is the estimated time necessary to run SA for a chain of length i, which can be found in Table 4.2, $m = 50$ is the number of SA runs used to calculate performance, $n_r = 3$ is the number of relations and $n_o = 2$ is the number of operators.

This value will be useful in later experiments, discussed in Chapter 5, to relate the running time of HHs to the number of chain checks that could be performed in the same time.

4.1.3 Baseline Performance and Effective Chains

In the next experiment, we aim to determine whether there are chains that significantly enhance performance compared to using only the bit flip operator. To establish if the performance improvement is statistically significant, we will examine the confidence interval of the difference between the performance of chains (measured over 50 SA runs) and the performance of the bit flip operator. In Section 4.1.1, the performance of the bit flip operator was determined to be $\overline{f_0} = 50.02$ over 100 SA iterations. A strictly positive confidence interval indicates a significant improvement, allowing us to conclude that the chain enhances performance.

Instead of calculating the confidence interval for each chain individually, we aim to simplify the process by determining a baseline performance. This baseline performance represents the minimum performance a chain must achieve to be said to significantly improve performance. We assume the standard deviation of the objective values is at most $s_{max} = 4.93$ and that these values follow a normal distribution. Based on detailed calculations in Appendix A, we set the baseline performance to $f_b = 51.71$.

In future experiments, we define *effective chains* as those achieving a performance above $f_b = 51.71$. These effective chains are all chains which significantly enhance the performance of SA when included. The ability of different HHs to effectively discover such effective chains is crucial. For this reason, the set of effective chains will also play a role in evaluating the effectiveness of various HHs in subsequent experiments, discussed in Chapter 5.

4.2 Exploration of Chains of Maximum Length 4

In the next chapter, we will run several experiments and test the efficiency of different HHs for specific Key Performance Indicators (KPIs). Many of these KPIs require the performance of chains to be known. One possible tactic could be to evaluate the performance of all relevant chains for each experiment. However, since several chains will occur frequently in different experiments it is more time-efficient to determine the performance of all necessary chains beforehand. This helps optimize the time needed for all experiments and avoids double work. In addition to this, it gives us more insight into the expressive power of the concept of chains.

We elect to determine the performance of all chains of maximum length 4 constructed from basis operators $\Theta = \{\varphi, \varphi_1\}$ as described in Section 2.3 and relations $\mathcal{R} = \{RM, CM, BT\}$ as described in Section 2.3.3. The maximum length was set to 4 because there exist 3108 chains of this maximum length. This number is high enough so that finding effective chains can be sufficiently hard, but not so much that testing all chains is completely intractable. If we were to test all chains of length 5 with 50 SA runs, then this would already take an estimated time of 90 hours.

We run the Priority Tree Search algorithm with chain length 4 and $max\_to\_test = 3108$ to determine the performance of all chains. For each chain, the performance is determined according to Algorithm 3, with $m = 50$, as determined in Section 4.1.

4.2.1 Overview

Table 4.3 summarizes the number of effective and normal chains found for different chain lengths. Notably, only a small section of chains significantly improves the search quality (only 4.05%). The existence of these effective chains is a promising indication of the potential value of chain structures. At the same time, the fact that only a small proportion of chains are effective suggests that HHs are necessary to discover them efficiently.

Maximum Chain Length	**1**	**2**	**3**	**4**
Number of effective chains	1	6	33	126
Total number of chains	12	84	516	3108
Percentage effective chains (%)	8.33	7.14	6.40	4.05

Table 4.3: Overview of the number of normal and effective chains for different maximum chain lengths for $f_b = 51.71$ and $m = 50$.

Figure 4.1 shows the percentage of chains that meet a minimum performance for different chain lengths. We note that as the chain length increases it seems that a smaller percentage of chains can improve the performance. Given that many more possible chains exist for larger chain lengths, this means it is harder to find such chains.

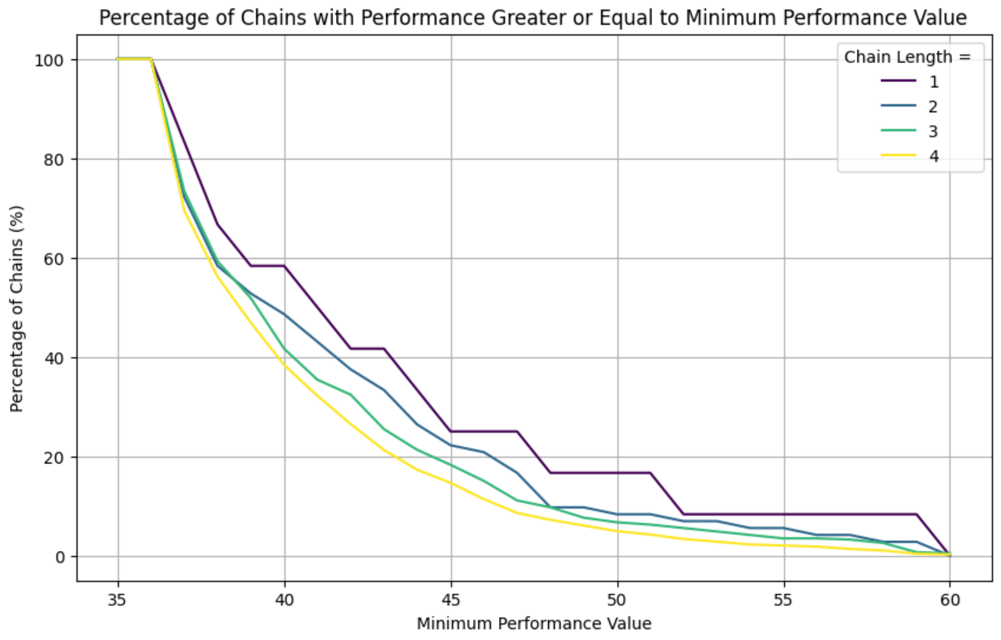


Figure 4.1: Percentage of chains of maximum length 4 that meet a minimum performance.

The histogram in Figure 4.2 shows the number of chains that exist within performance buckets of size 1. Notably, the vast majority of chains decrease the performance of SA, compared to the performance when only the bitflip operator is used, $\overline{f_0} = 50.02$.

The minimum performance observed is 36, which is the value associated with all trapdoor rooms containing 1's. Note that SA with the bitflip operator can reach this state from any state where every room contains at least one 1 bit, with only strictly improving steps. This means that the state with only 1's is relatively easy to find by the SA algorithm, especially at low temperatures. Since the bitflip operator is always included when performance of a chain is evaluated, this explains why no lower performance value was observed.

The maximum value observed is 60, which is the value associated with all trapdoor rooms containing only 0's. A performance value of 60 means that over 50 SA runs the optimal value was always found. The fact that chains exist which achieve this performance value suggests that the chain structure can be useful in defining new operators. Of the 8 chains that have a performance of 60, 6 chains have a length of 4 and 3 chains have a length of 3. This suggests that longer chains have more expressive power than shorter chains, and can thus achieve higher performance. This motivates the need for HHs which determine which chains to evaluate first.

4.2.2 Short Chains

To gain more insight into the performance of specific chains we consider the highest performing chains of fixed length. Table 4.4 and Table 4.5 show the performance for all chains of length 1 and 2 respectively. A detailed list of all chains and their performance can be found in the Appendix D. Recall from Section 4.1 that the 95% confidence intervals of these performance values have a width of around 3. This means that the ranking induced by the performance of chains may be subject to changes if more runs were performed.

For chain length 1, the chain with the highest performance is

$$\varphi_1 \xrightarrow{BT} \varphi,$$

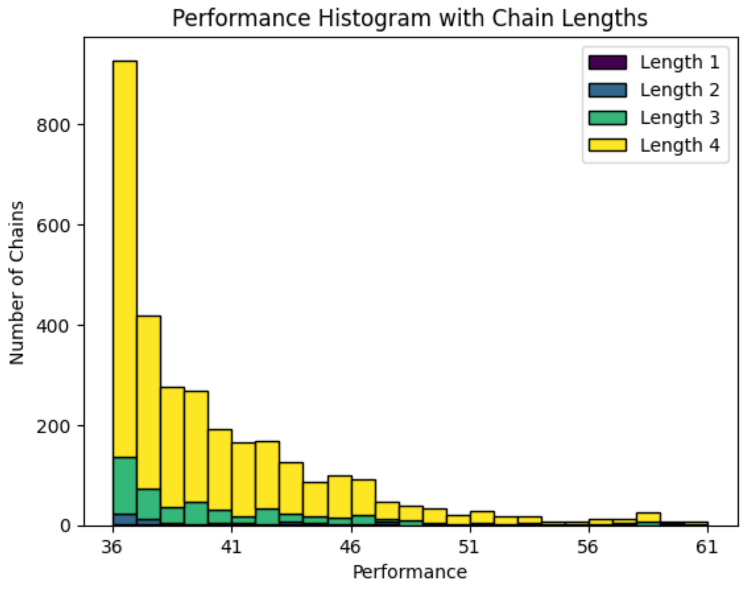


Figure 4.2: Histogram of chains of various length and their performance.

which selects a random bit and makes it 0. The second-best operator is

$$\varphi \xrightarrow{CM} \varphi,$$

which selects two random bits in the same room and flips them. All other chains have a lower performance value than the bitflip operator, at $\overline{f_0} = 50.02$.

It is also worth noting that given the relation and basic operators included, it is possible to create chains that do nothing new. For example, the

$$\varphi_1 \xrightarrow{BT} \varphi_1$$

chain selects a bit, flips it to 1, and then selects the same bit and flips it to 1 again. This is the same as the φ_1 operator does. Another example is the

$$\varphi \xrightarrow{BT} \varphi$$

chain. This chain selects a bit and flips it, it then selects the same bit and flips it back. This leaves the current solution unchanged. When such chains are tested in SA they have the effect of using the number of iterations we have without exploring further. This explains why their performance is also generally low.

When looking at the performance of chains of length 2, we note that several of these chains also contain subchains which effectively do not change the solution. For example, the highest-performing chains of length 3,

$$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$$

and

$$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$$

have the same effect as the φ_0 operators but in 3 iterations instead of 2.

In future research, it may be useful to disregard these types of chains completely. However, for all mining-based HHs, chains are constructed based on observations. Since these chains may still be observed, it is also likely that they are discovered by the algorithm. It thus does not make sense to disregard these types of chains completely in evaluating the effectiveness of these HHs. Future alterations to these HHs may add a postprocessing step to filter out such chains. However, due to time constraints this was not explored further in this research.

Chain	Performance
$\varphi_1 \xrightarrow{BT} \varphi$	59.92
$\varphi \xrightarrow{RM} \varphi$	51.10
$\varphi \xrightarrow{CM} \varphi$	47.68
$\varphi \xrightarrow{BT} \varphi$	44.74
$\varphi_1 \xrightarrow{CM} \varphi$	43.40
$\varphi \xrightarrow{CM} \varphi_1$	41.60
$\varphi_1 \xrightarrow{RM} \varphi$	40.20
$\varphi \xrightarrow{RM} \varphi_1$	38.10
$\varphi_1 \xrightarrow{BT} \varphi_1$	37.70
$\varphi \xrightarrow{BT} \varphi_1$	37.20
$\varphi_1 \xrightarrow{RM} \varphi_1$	36.72
$\varphi_1 \xrightarrow{CM} \varphi_1$	36.40

Table 4.4: Performance of chains of length 1.

Chain	Performance
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	59.52
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	59.28
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	57.36
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	55.44
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	53.16
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	51.24
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi$	49.62
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi$	47.68
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi$	47.46
$\varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	47.42
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi$	47.38
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	47.22
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	46.98
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	46.60
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	46.18
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	45.42
$\varphi_1 \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi$	44.82
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1$	44.56
$\varphi_1 \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	44.08

Table 4.5: Performance of chains of length 2.

4.2.3 Important Chains

Within the defined set of chains, there are a number of chains which we expect to have a considerable effect, based on our knowledge of the Trapdoor problem. The first is the

$$\varphi_0 := \varphi_1 \xrightarrow{BT} \varphi$$

chain. As discussed previously, this chain selects a bit and flips it to 0. Since the optimal solution contains only 0s, we expect this chain to have a considerable effect.

The next chain which we expect to have a considerable effect is the

$$\varphi_0^2 := \varphi_0 \xrightarrow{RM} \varphi_0 = \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$$

chain. This chain consists of two φ_0 chains connected by a room relation. This means that the chain selects two bits randomly in the same room (can be the same bit), and flips them to 0.

The performances of the φ_0^2 and φ_0 chains can be found in Table 4.6. Given that there is still some randomness in the performance of chains, these (high) performance values align with what we would expect. It is also worth noting that many of the top-performing chains contain parts of either φ_0 or φ_0^2. Some other chains do essentially the same as φ_0 or φ_0^2 but are formatted differently. Examples are

$$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi,$$

which has a performance of 59.52, and

$$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi,$$

which has a performance of 60.

This further strengthens our claim that the φ_0^2 and φ_0 chains are the most relevant chains to discover for a maximum length of 4. In the next experiments, we will pay special attention to whether each HH discovers these chains and how long this takes.

Name	**Chain**	**Performance**
φ_0^2	$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	60.00
φ_0	$\varphi_1 \xrightarrow{BT} \varphi$	59.92

Table 4.6: Performance of φ_0^2 and φ_0 chains.

Chapter 5

Hyper-Heuristic Experiments

In Chapter 4, we have tested the performance of chains up to length 4. We found that several chains have a higher performance than the `bit_flip` operator. This means that by including these chains we, on average, find a better solution, using the same Simulated Annealing (SA) algorithm.

However, in practical settings, there may not be enough computational time available to test all chains and select the best one. To this end, we have developed several Hyper-Heuristics (HHs) which return an order in which to test these chains. In this chapter, we discuss the results of several experiments performed to test the effectiveness of the different HHs developed for the Trapdoor problem.

In section 5.1, we discuss the Key Performance Indicators (KPIs) used to evaluate the effectiveness of the different HHs.

In Section 5.2, we apply Priority Tree Search (PTS) to test chains in an ordered way. Using the results from the previous experiment we evaluate how effective this ordering is for finding higher-performance chains earlier. In some cases, we find that the ordering induced by PTS is not better than random sampling chains and testing them. We also find that the basis assumption of PTS, sub-chains of effective chains are also effective chains, does not hold.

In Section 5.3, we test the Mining Hyper-Heuristc (M-HH) for several values of len_sub. We find that in all cases a statistically significant positive relation exists between support values and performance values of chains. We also find that important chains are discovered early. However, for lower values of len_sub fewer chains are discovered, while for higher values of len_sub the algorithm is much slower.

In Section 5.4, we test the Population Mining Hyper-Heuristic (PM-HH) for $len\_sub = 15$. We find that the algorithm does find more chains than the mining case with one generation. However, in this case, no positive relation could be discovered between the support value of a chain and its performance. This is likely the result of the limited definition used for good sub-sequences.

Finally, we compare the three HHs developed across KPI's. We find that depending on the computational time available, different HH approach are appropriate.

5.1 Key Performance Indicators (KPIs)

Each of the HHs developed in this thesis returns not simply a single chain operator but an ordered list of chain operators. In assessing the effectiveness of HHs we will work under the assumption that there is a limited amount of computational time available. This means that likely, not all chains on this ordered list can be evaluated in terms of their performance. An effective HH thus returns an ordered list of operators for which high-performance operators are consistently ranked higher than low-performing operators. In addition to this, it is important that generating this list does not take up too much time, as this takes away from the time available for testing chains. Finally, it is relevant whether all possible chains are featured on this list, and especially if effective chains appear in this ordered list.

In evaluating the effectiveness of the different HHs developed for the Trapdoor problem, we will consider several key performance indicators (KPIs) which summarize these qualities. These KPIs are distinct from the performance of an individual chain and are used to assess the overall efficiency of the HHs. The KPIs used in this study are outlined in detail below.

5.1.1 General KPIs

Time Necessary to Generate Ordered List of Chains

For each HH discussed we will measure the time taken by the HH to generate an ordered list of chains. This time is distinct from the time needed to test chains present in the ordered list. If a lot of time is necessary to generate the ordered list, then less computational time is available to test the chains in the ordered list. This KPI is relevant for all HHs.

We express the time taken both in seconds and in the equivalent number of chain checks that could be performed in the same amount of time. Time in seconds can be converted to time in chain checks by dividing it by the average time needed to perform a chain check. This is equal to 17 seconds, as calculated in Section 4.1.2.2. We choose to also display time in terms of chain checks as this allows us to more easily compare time necessary for the generation of an ordered list of chains, and the time taken to test the chains in the list.

Reach

We find that in some cases HHs return a list in which not all chains appear. This means no information is available on when to test these chains, and they will likely only be tested once all chains on the ordered list have been tested. To account for this, we measure the proportion of the total possible chains (up to a specified length) that appear in the ordered list returned by the HH.

Since we only consider chains of at most length 4 consisting of 2 basis operators and 3 relations, the total number of possible chains is equal to 3108. Let C be the number of unique chains that appear in an ordered list returned by an HH. The *reach* of this HH is then calculated as $C/3108$. This KPI is particularly relevant to M-HH and PM-HH, which do not necessarily discover all chains.

Recall

The goal of testing chains in the ordering is not only to discover the performance of each chain but, more importantly, to find effective chains (which improve the SA search) efficiently. It is thus also relevant how many effective chains appear in the ordering. We measure the proportion of chains with performance exceeding a baseline threshold of $f_b = 51.71$ that are included in the ordering returned by HHs.

In Section 4.2, we found that there are 126 chains that meet the baseline performance of $f_b = 51.71$. Let E be the number of unique effective chains that appear in an ordered list returned by an HH. The *recall* of this HH is then calculated as $E/126$. This KPI is particularly relevant to M-HH and PM-HH, which do not necessarily discover all chains.

5.1.2 Order-Related KPIs

To assess the quality of the orderings induced by each HHs we define additional KPIs to measure desirable qualities of a good ordering of chains. These KPIs require knowing the performance of the chains that appear in the ordering. To save time, we will use the performance values found in Section 4.2 instead of recalculating the performance for each ordered list of chains.

Testing the chains in the ordering takes time. To measure this, we use the number of chains tested rather than the time in seconds. This approach is chosen because the testing time can vary based on the number of annealing runs required, which differs across problems due to the variance in solution quality. Additionally, the time needed for a single annealing run can differ depending on the problem size. Using the number of chains tested as a metric allows for consistent comparison across different HHs.

Position of Important Chains

A good ordering of chains should prioritize the most effective chains at the top. To assess this, we measure the positions of specific known effective chains (such as φ_0 and φ_0^2 defined in Section 4.2.3) in the ordering of chains produced by the HH. This corresponds to how many chains must be checked before the effective chain is discovered. This KPI is relevant to all HHs.

Quality of Ordering

Furthermore, a good ordering of chains should have higher-performance chains positioned earlier in the list. This allows these high-performance chains to be tested and discovered sooner.

To measure this, we introduce a new variable $C(t, f_{min})$, which represents the number of chains discovered after t chain checks that meet or exceed the performance threshold f_{min}.

The variable $C(t, f_{min})$ is calculated as follows: First, we set a minimum performance value, denoted as f_{min}, which is the threshold that chains must meet or exceed to be considered high-performing. Next, we fix the number of chains to test, represented as t. For the given ordering, we calculate $C(t, f_{min})$, the number of chains among the top t that have a performance value of at least f_{min}. This calculation is repeated for various values of t and f_{min}.

For each heuristic (HH), we plot the value of $C(t, f_{min})$ for t ranging from 1 to 3108 and various values of f_{min}. The x-axis represents t, the number of chains tested, and the y-axis represents $C(t, f_{min})$, the number of chains with performance at least equal to the performance threshold. Different values of f_{min} are represented by different lines on the graph.

A heuristic with a higher quality ordering will have a higher value of $C(t, f_{min})$ for a given t and f_{min}, indicating that fewer tests are needed to find effective chains.

5.1.3 Support-Related KPIs

Relationship Support and Performance

Several HHs induce on ordering based on the support values of the chains discovered. The underlying assumption in this step is that the support value is a good predictor of the performance of a chain. To examine this we measure the correlation between the support value of a chain and the measured performance of a chain. This KPI is relevant for M-HH and PM-HH, as it validates the reliability of support values as predictors of chain performance.

5.2 Experiment 1 - Priority Tree Search

In calculating the ground truth values as discussed in Section 4.2, we also ran a Priority Tree Search (PTS) with a maximum number of tries equal to the number of chains of at most length 4 (i.e. 3108). This ensures that all chains are discovered eventually. This results in an ordering of chains, as they were discovered during the algorithm. This ordering is relevant as it gives us some idea of how long it would take to discover certain chains. If the running time is limited one can imagine that only the first few chains can be tested at all. This section is dedicated to discussing this ordering.

We find that with enough running time, the algorithm eventually tests the performance of all chains up to some maximum length, meaning it provides a total reach of the chain space as well as total recall. The method is time efficient, in terms of time needed to generate an ordering. Almost no additional time is necessary for anything other than performance checks. However, the ordering induced by the algorithm does not effectively prioritize high-performing chains. This can be explained by the fact that the assumption underlying the algorithm does not hold: sub-chains of high-performance chains, are not necessarily high-performance themselves.

5.2.1 Experimental Setup

We test the PTS approach for Trapdoor parameter settings as outlined in Section 2.1.1, SA parameter settings as outlined in Section 2.2.2.3 and PTS parameter settings as outlined in Section 3.2.2.

5.2.2 Time

Strictly speaking, in PTS, the time needed to generate an order cannot be seen as separate from the time needed for testing the chains in this order. This is because the resulting performance values are used in determining the order of future chains to test. However, for comparison with other HHs, it may still be useful to gain some insight into how much time is needed for the algorithm if chain checks are disregarded. To this end, we can run the algorithm again while now looking up performance values for each chain from a dictionary containing all performance values. This means almost no extra time is spent on evaluating chains. The algorithm then takes only 0.04 seconds. Compared to the on average 0.35 seconds needed for a single chain check, this means that the underlying heap structure almost takes no additional time.

5.2.3 Quality of Ordering

Figure 5.1 visualizes the quality of the ordered list induced by the PTS heuristic. This list includes all chains up to a length of 4, arranged in the order in which they are discovered by PTS. The visualization helps to assess how effectively PTS prioritizes higher performance chains early in the list.

In this figure, we see that it takes at least 1500 chain checks before all chains with minimum performance of 59 and 60 have been discovered. For the chains of minimum performance 53, 55 and 57 this takes almost 2500 chain checks. Finally, at least 2700 chain checks are needed to discover all effective chains.

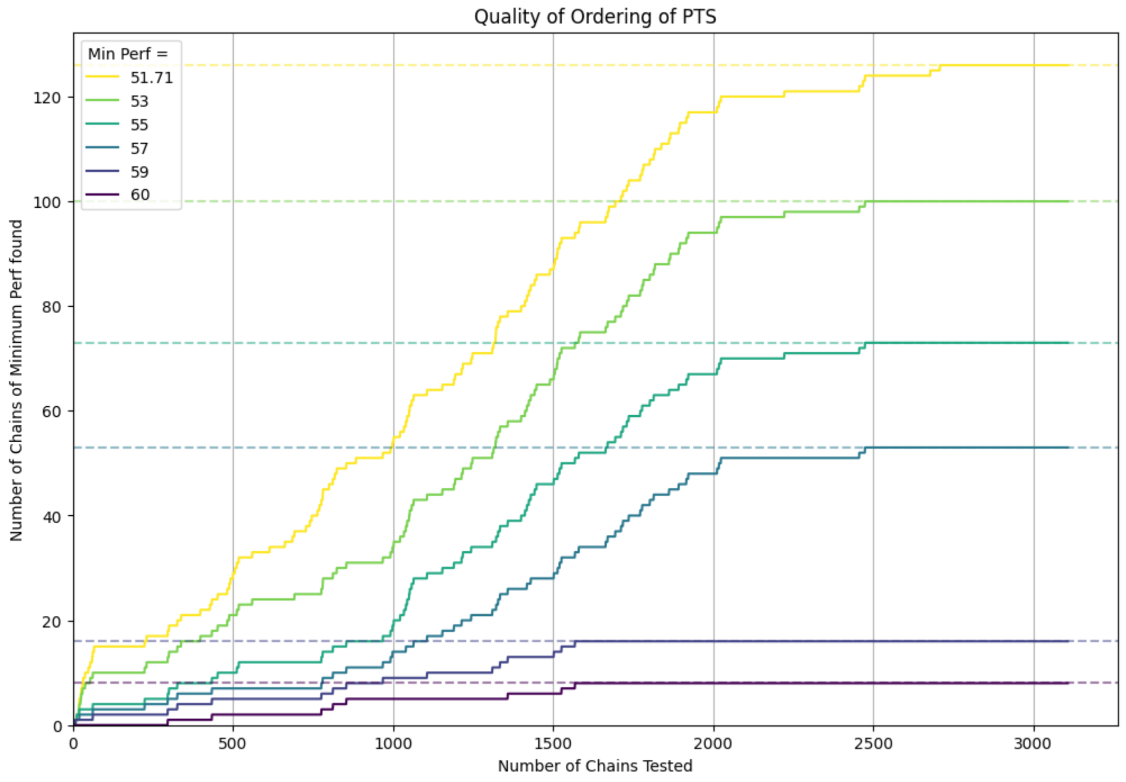


Figure 5.1: Quality of Ordering for PTS for different values of min_perf.

To gain a deeper understanding of what these numbers mean, we compare this to what we expect to happen if we randomly sample chains of maximum length 4, and test them. Figure 5.2 shows the expected number of chains found for the number of chains tried, for both random sampling and PTS. The slopes of the lines are determined by dividing the number of chains that meet the minimum performance by the number of chains of maximum length 4 (3108).

For $min\_perf = 51.71$, we see that the PTS algorithm is expected to find more chains than if we randomly sample chains. However, for $min\_perf = 56$ the PTS performs worse than random sampling if we can only check 200 to 1000 chains. For $min\_perf = 60$, it is also more efficient to randomly select chains if we only have time to run less than 300 chains.

These results suggest that PTS is not very efficient at finding high-performance chains. This motivates the need for other HH approaches.

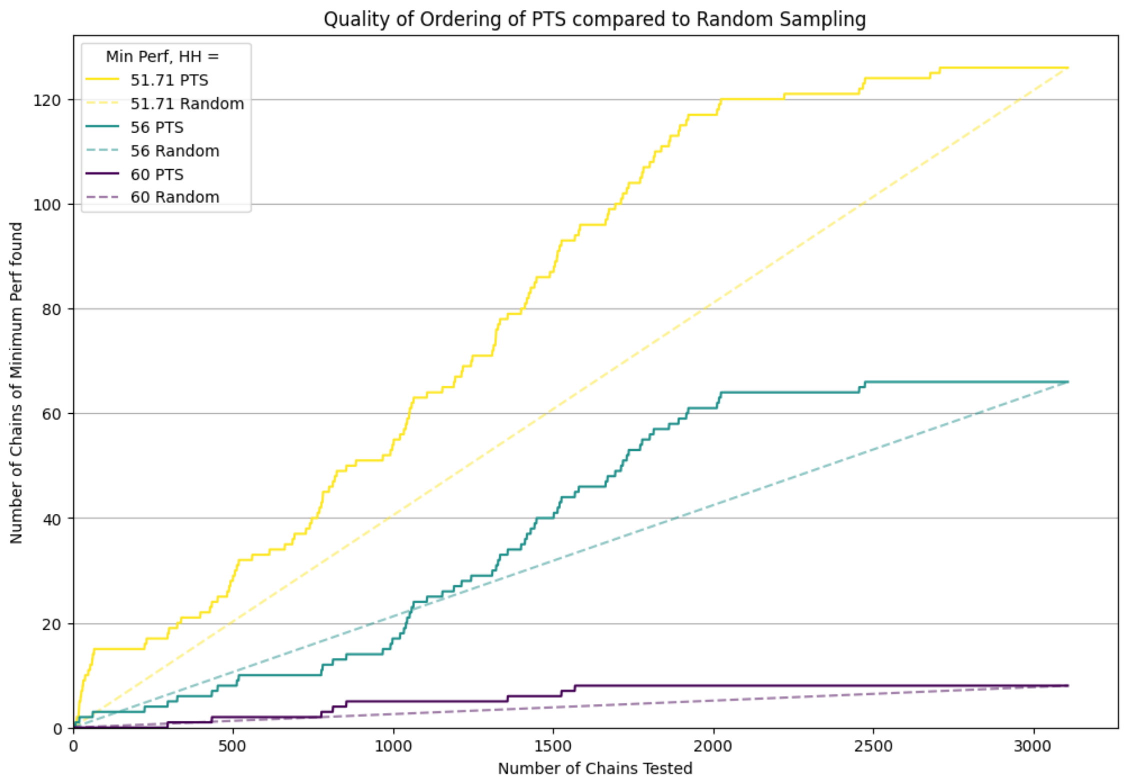


Figure 5.2: Quality of Ordering for PTS compared to Quality of Ordering for randomly sampling chains.

5.2.4 Important Chains

Table 5.1 shows the position of some relevant chains as they are discovered in PTS. A more detailed list of all chains can be found in the Appendix.

We see that φ_0 is tested very early, after 9 chains already. This makes sense, as it is a relatively short chain, meaning it occurs sooner in the chain tree. However, the φ_0^2 chain is only tested (and thus discovered), after 777 chains have been tested. This is relatively late, especially since there are only 516 chains of this maximum length. We can explain the reason for this by looking at subchains of φ_0^2 (also shown in the table). To reach the node corresponding to φ_0^2, we will first need to expand the node corresponding to its left subchain

$$\varphi_L = \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1.$$

We see that this chain is also tested soon, after 18 chain tests. However, its performance is very low: 37.08. With such a low performance value the chain likely stays at the bottom of the heap for a very long time. We can conclude that our assumption that sub-chains of effective chains are also effective chains clearly does not hold. Such a low performance of φ_L also suggests that even though PTS is a probabilistic HHs, it is likely consistently bad at discovering the φ_0^2 chain. This, in part, is the motivation for the development of the other HHs discussed in this paper.

5.3 Experiment 2 - Mining Hyper-Heuristic

In this section we evaluate the effectiveness of an alternative approach: the Mining Hyper-Heuristic (M-HH), which aims to exploit more information from a single annealing run, so that fewer annealing runs are necessary.

Based on the experiments discussed in this section, we find that the support values for a large portion of the chain space of size 3108 can be calculated relatively fast, in at most the time needed to perform 42 chain performance checks. We find a statistically significant positive relationship between the support and

Name	Chain	Performance	Position
φ_0^2	$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	60.00	777
φ_0	$\varphi_1 \xrightarrow{BT} \varphi$	59.92	9
φ_R	$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	57.36	1041
φ_L	$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1$	37.08	18

Table 5.1: Overview of position of relevant chains for PTS.

performance of a chain operator, suggesting that support is a good indication of the actual performance of a chain. By sorting the chains from high to low support values we can find on order in which to test the performance of different chains. This ordering consistently discovers high performance chains early on. However, many chains with high performance are never discovered, because longer chains only appear in short sub-sequences very rarely. We may increase the length of sub-sequences to discover more chains, but this comes at the cost of increasing mining time.

5.3.1 Experimental Setup

We test the M-HH approach for Trapdoor parameter settings as outlined in Section 2.1.1, SA parameter settings as outlined in Section 2.2.2.3 and M-HH parameter setting as outlined in Section 3.3.5.

To determine the appropriate number of sequences for effective mining with varying parameters, we must balance reliability and computational efficiency. A higher number of sequences generally leads to more reliable support values, though it increases the computational cost. We expect the computational cost to increase linearly due to more sequences being generated, and the fact that more sub-sequences exist to be mined for chains.

For this study, we set the number of sequences to 500. This number ensures that the mining technique remains competitive with the PTS technique. Generating one sequence roughly equates to a single run of the SA algorithm. Thus, generating 500 sequences is expected to take approximately the same time as checking the performance of 10 chains.

While the mining algorithm itself is deterministic, the sequences produced by Simulated Annealing may vary due to the inherent randomness of the algorithm. To assess the consistency of our results from a single mining iteration, we conduct 20 experiments, each involving the generation of sequences and subsequent mining for chains. This will help us understand the variance in outcomes across different runs.

For each sequence, we perform the mining algorithm for sub-sequence lengths of 5, 10, 15, 20, 25, 30 and 35 and maximum chain lengths of 3 and 4.

5.3.2 Reach and Recall

Table 5.2 summarizes the average reach and recall with standard deviations for different sub-sequence lengths and a maximum chain length of 4. Recall that the number of chains found is the number of chains for which a support value is calculated and the support value is only calculated if a chain is encountered at least 20 times in good sub-sequences.

We see clearly that the number of found chains (effective and normal) increases with the sub-sequence length. This makes sense, as more chains may be contained in a longer sub-sequence than in a shorter sub-sequence. It is important to realise that we only expect chains that end with operator φ to appear in good sub-sequences. This is because good sub-sequences end with a lynchpin, as do all chains contained in the sub-sequence. The lynchpin of a good sub-sequence corresponds to an operator which brings about a maximal change in objective value and flipping a bit to 1 can never bring about this large change. This means that the mining technique can only discover half of all chains, meaning the maximum value of reach we can achieve is 50%. However, even for a sub-sequence length of 35, the reach is still only 46.40% (± 0.53) meaning that this maximum value has not been achieved.

Incidentally, all effective chains do end with the φ operator, so theoretically it is possible for M-HH to discover all effective chains, and achieve a recall of 100%. The highest recall occurs at a sub-sequence length

Sub-sequence length	**5**	**10**	**15**	**20**	**25**	**30**	**35**
Average Reach (%)	0.68	3.45	15.60	31.56	40.75	44.83	46.40
Std Dev	0.01	0.21	0.54	0.26	1.21	0.22	0.53
Average Recall (%)	0.79	4.09	13.53	29.21	51.00	62.50	71.23
Std Dev	0.00	0.29	0.30	1.25	6.48	1.33	5.36

Table 5.2: Summary of reach (%) and recall (%) M-HH for $len\_chain = 4$, $f_b = 51.71$ and $num\_seq = 500$ for different values of len_sub. Average and standard deviation are calculated over 20 iterations.

Sub-sequence Length	**5**	**10**	**15**	**20**	**25**	**30**	**35**
Time sequence generation							
Average (s)	282.84	282.84	282.84	282.84	282.84	282.84	282.84
Std Dev (s)	41.66	41.66	41.66	41.66	41.66	41.66	41.66
Average (cc)	16.64	16.64	16.64	16.64	16.64	16.64	16.64
Std Dev (cc)	2.45	2.45	2.45	2.45	2.45	2.45	2.45
Time mining							
Average (s)	1.67	4.76	21.90	92.83	110.65	217.04	424.73
Std Dev (s)	0.28	0.67	21.52	60.57	3.03	6.31	29.54
Average (cc)	0.10	0.28	1.29	5.46	6.51	12.77	24.98
Std Dev (cc)	0.02	0.04	1.27	3.56	0.18	0.37	1.74
Total time							
Average (s)	284.51	287.60	304.74	375.67	393.50	499.88	707.57
Std Dev (s)	41.71	42.07	45.07	59.18	41.76	38.79	41.49
Average (cc)	16.74	16.92	17.93	22.10	23.15	29.40	41.62
Std Dev (cc)	2.45	2.47	2.65	3.48	2.46	2.28	2.44

Table 5.3: Summary of time used by M-HH in seconds (s) and chain checks (cc) for $len\_chain = 4$ and $num\_seq = 500$ for different values of len_sub, averaged over 20 iterations with standard deviations.

of 35, at 71.23% (± 5.36). This means the mining technique still fails to find a large proportion of effective chains.

5.3.3 Time

Table 5.3 summarizes the time needed in M-HH for different phases, as well as the total time taken to generate an ordering. The total time spent consists of the time used to generate sequences (consistent at around 282.84 (± 41.66) seconds) and the time spent mining chains (variable for different parameter settings).

We find that the time it takes to generate 500 sequences then takes roughly the same amount of time as it takes to test around 16.64 (±2.45) chains. This value is slightly higher than the 10 chain checks we would expect it to take. This is likely because during sequence generation we not only run SA but also log all steps taken along the way.

The total time needed ranges between 284.51 (±41.71) seconds to 707.57 (± 41.49) seconds. This corresponds to the time it takes to test between 16.74 and 41.62 chains respectively. These relatively low values suggest that the more time-consuming part of the mining process is testing the chains in order of their support values. Details on this can be found in the next sections.

Next, let us have a closer look into the effect of different parameters on the time spent mining. Figure 5.3 visualises the total time taken to mine chains for different values of sub-sequence length as well as different values of chain length. For a maximum chain length of 4, the time spent mining seems to increase exponentially with the length of sub-sequences. This is to be expected, as the number of chains in a sub-sequence also increases exponentially with sub-sequence length. For a chain length of 3, this effect is less

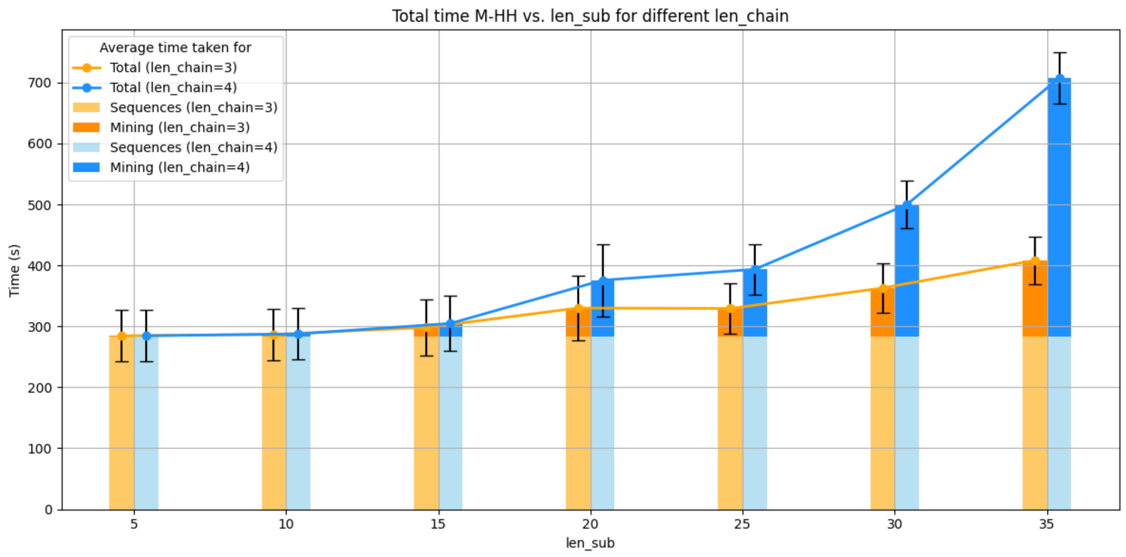


Figure 5.3: Time in seconds for different phases of M-HH for different values of *len_chain* and *len_sub* averaged over 20 iterations. The standard deviation of the total time is shown in black brackets.

dramatic. This makes sense, as eventually chains will have reached the maximum length and can no longer be elongated. Once this point is reached we expect the time spent mining to increase linearly with the length of sub-sequences. This can potentially already be seen for a chain length of 3.

5.3.4 Relationship Support and Performance

Once the mining process is finished it returns a list of chains found and for each chain a corresponding support value. This support value is an indication of how much more likely a chain is to appear in a good sub-sequence as opposed to a normal sub-sequence. In this section, we look into the relation between this support value and the measured performance of a chain.

Figure 5.4 shows the relationship between support (Sup) and Performance (Perf) for a single iteration of mining with $len\_sub = 35$. The data points are plotted alongside the regression line, which is represented in red. The linear regression equation is given by:

$$Perf = 36.85 + 2.32 \cdot Sup$$

The slope of the regression line is 2.32, indicating that for each unit increase in support, performance increases by approximately 2.32 units. The intercept is 36.85, meaning when support is zero, performance is expected to be 36.85.

The p-value for the slope is significantly low (< 0.05), suggesting that the relationship between support and performance is statistically significant. The low p-value indicates that the null hypothesis (which states that there is no relationship between support and performance) can be rejected with high confidence.

We created such a scatter plot and regression line for all 20 iterations of mining and for all values of *len_sub*. The average results of this analysis can be found in Table 5.4. Specific results per iteration can be found in Appendix B. For all values of *len_sub* we find a positive slope, indicating there is a positive relation between support and performance. The R-squared value is at least 0.37 in all cases, meaning that at least 37% of all variance is explained by this relation. In all cases, p-values are low (< 0.05) indicating that the relation is statistically significant. The slope values show a general increasing trend with higher sub-sequence length, suggesting a stronger relationship between support and performance as more data points are included. The R-squared value also increases with higher sub-sequence length, indicating that more variance is explained.

The statistically significant positive relationship between support and performance suggests that checking chains in order from high support to low support is likely to lead to the discovery of more high-performance chains in less time. This ordering is further discussed in the next sections.

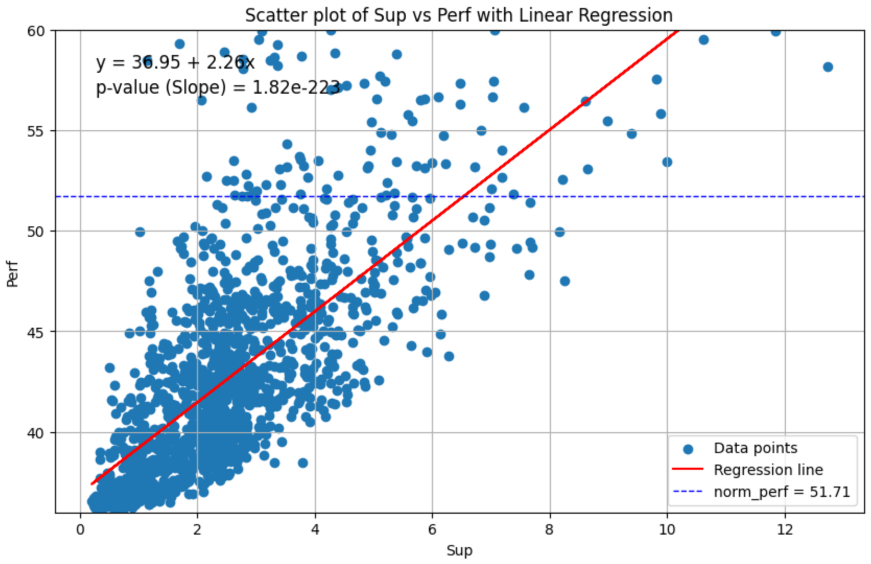


Figure 5.4: Scatterplot of support and performance of chains discovered by M-HH with a sub-sequence length of 35 for one iteration.

Sub-sequence length	**Intercept**	**Slope**	**P-value Intercept**	**P-value Slope**	**R-squared**
5	40.05	1.78	0.00	0.00	0.37
10	38.54	1.82	0.00	0.00	0.45
15	38.08	1.59	0.00	0.00	0.42
20	37.96	1.69	0.00	0.00	0.45
25	37.59	1.95	0.00	0.00	0.47
30	37.21	2.11	0.00	0.00	0.49
35	36.85	2.32	0.00	0.00	0.48

Table 5.4: Average regression results for the relationship between support and performance for M-HH with different values of *len_sub*, calculated over 20 iterations.

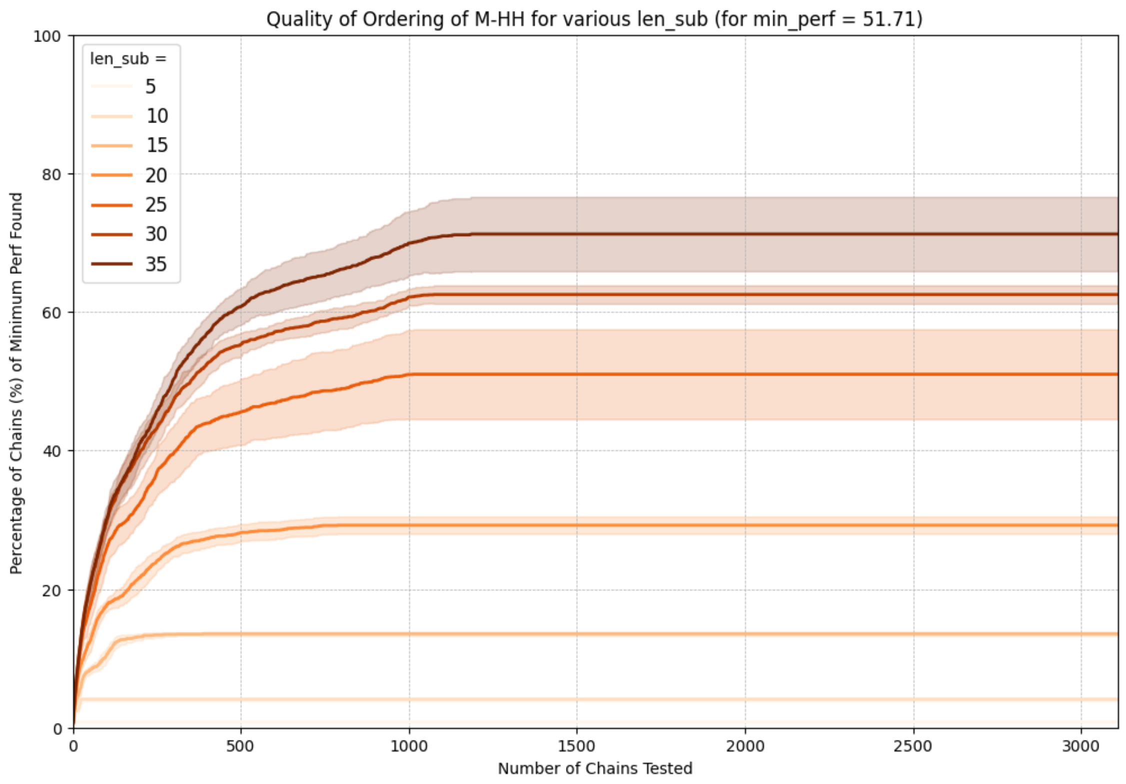


Figure 5.5: Quality of Ordering of M-HH with different values of len_sub for $min\_perf = f_b$ averaged over 20 iterations. The standard deviation is shown by the shaded area around the curve.

5.3.5 Quality of Ordering

To visualise the quality of the ordering induced by support value, we plot the average number of effective chains found, over the number of chains that have been tested for different sub-sequence lengths in Figure 5.5. Variance over the 20 iterations of M-HH is shown by the shaded area (± std). Notably, for sub-sequence lengths of 25 or 35 a larger variance is observed, than for other sub-sequence lengths. This suggest that for these lengths there exist several chains which are discovered sometimes, but not consistently, likely because the frequency with which they occur is close to the minimum frequency.

We observe a clear pattern of more effective chains being found for higher len_sub values. In addition, we find that len_sub dominates all other parameter setting in terms of the number of effective chains found for any number of chains tried.

For this reason we further examine this case, M-HH (35), in terms of how efficiently chains are discovered of some minimum performance. These results are shown in Figure 5.6. A similar pattern is observed for all minimal performance values; almost immediately several chains are found that meet the minimum performance. After around 1400 chain checks all discovered chains have been tested, but not all chains that meet the minimum performance have been found. This can be observed in the difference between the dotted line and normal lines of the same colours at $x = 1400$. Later, in Section 5.5, we will compare these results to the other HHs discussed.

Variance over the 20 iterations of M-HH (35) is shown by the shaded area (± std). We observe that the variance is relatively small for fewer chain checks, indicating that the method consistently discovers chains with high performance early, based on the ranking. We also find that the variance increases for a higher number of chain checks. This indicates that the algorithm is less consistent in terms of the number of high-performance chains it discovers in unlimited time. This variance is likely a result of some chains only appearing very rarely, and thus only being discovered in some iterations of M-HH but not others.

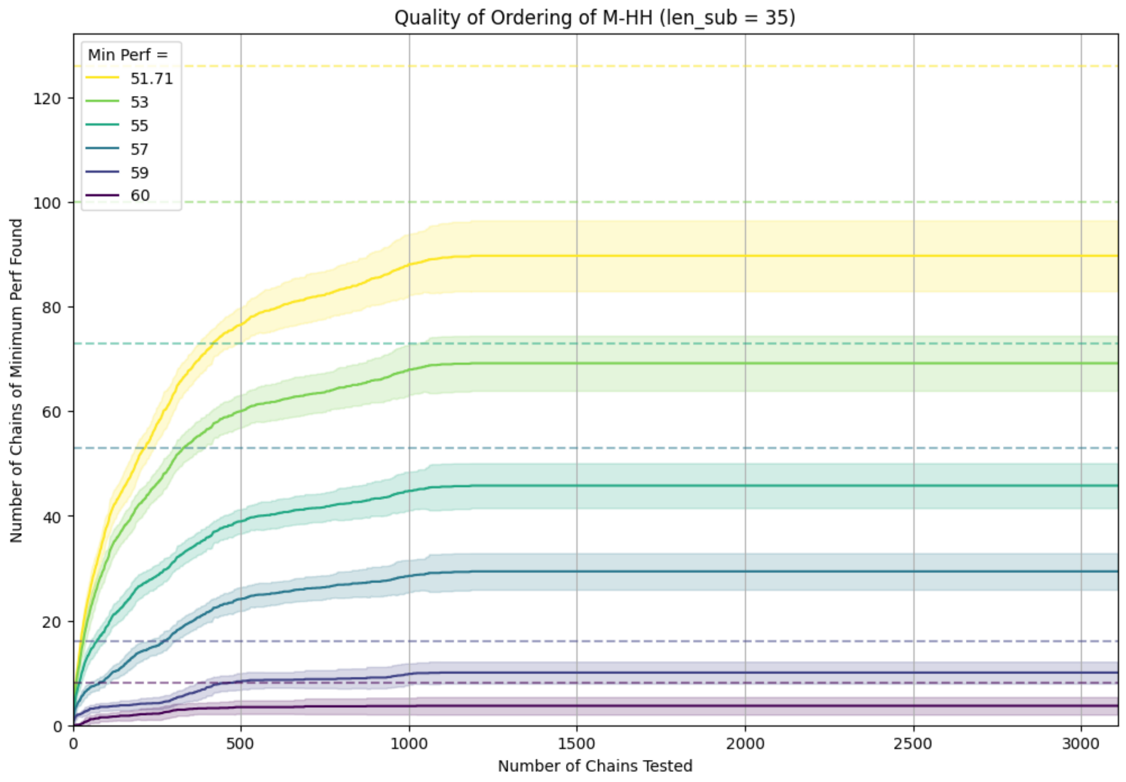


Figure 5.6: Quality of Ordering of M-HH ($len\_sub = 35$) for different values of min_perf averaged over 20 iterations. The standard deviation is shown by the shaded area around the curve.

5.3.6 Important Chains

Table 5.5 shows the average positions of some relevant chains in the order induced by support values (high to low). We see that φ_0 is found consistently, but its position increases with sub-sequence length. This is likely because more chains are found for longer sub-sequence lengths and some may have higher support values than φ_0 (for example the φ_R chain).

The φ_0^2 chain is only found for sub-sequence lengths above 20. This can be explained by the fact that this is a relatively long chain with a low probability of appearing randomly in a short sub-sequence. Once the chain is found it remains at a consistently high position in the ranking, even as more chains are discovered. This is a promising result, as the φ_0^2 chain is likely the most relevant chain to discover. The φ_L is not found for any sub-sequence length. Due to it ending with the φ_1, the mining technique can never discover it, at least for the current definition of good sub-sequences.

The fact that the φ_R chain is consistently ranked higher than the φ_0 chain, even though it performs worse, highlights a possible issue in the mining technique resulting from the definition of support. We expect the φ_R has such a high support value because the φ_0^2 often induces large changes in objective value and thus occurs often in good sub-sequences. As a result, its right sub-chain φ_R also occurs more often in good sub-sequences, even though it cannot be said that it causes these large changes in objective value. Future adaptations of the mining algorithm may aim to take this into account by giving some preference to longer chains over their sub-chains.

Chain		**5**	**10**	**15**	**20**	**25**	**30**	**35**
φ_0^2	Mean Position	-	-	-	23.16	28.15	39.10	42.10
	Std Dev	-	-	-	13.00	17.11	20.18	15.68
	Found(%)	0.00	0.00	0.00	95.00	100.00	100.00	100.00
φ_0	Mean Position	5.15	22.30	110.70	255.20	327.60	375.80	390.55
	Std Dev	0.36	2.30	12.95	35.50	37.13	50.66	49.43
	Found (%)	100.00	100.00	100.00	100.00	100.00	100.00	100.00
φ_R	Mean Position	-	5.35	26.50	64.55	88.80	114.45	122.85
	Std Dev	-	1.39	6.22	12.40	15.15	24.31	23.15
	Found (%)	0.00	100.00	100.00	100.00	100.00	100.00	100.00

Table 5.5: Summary of mean positions of relevant chains in ordering returned by M-HH for different sub-sequence lengths. For each chain the average position and standard deviation are calculated over 20 iterations. The percentage of iterations for which the chain is discovered is shown as well (Found).

5.4 Experiment 3 - Population Mining Hyper-Heuristic

Based on the previous experimental results, it seems that the mining technique is a promising approach. However, the technique does not discover all chains that improve performance, likely because longer chains only appear very rarely in a good sub-sequence. We may increase the number of chains found by increasing the sub-sequence length, but this comes at the cost of increasing running time.

In this section, we outline the experiments performed with an alternative technique designed to address some of these shortcomings: the Population Mining Hyper-Heuristic (PM-HH). In this method, we aim to discover more chains by ensuring that promising chains already appear in the sequences. We do this by running mining once and including the highest support chains directly as operators in the next generation of sequences.

This experiment is not meant to be a complete exploration of the population mining technique for all different parameters. We only aim to show that the approach is promising in terms of increasing the number of discovered chains. Many later alterations to the algorithm may still help improve it further, but were not explored due to time constraints.

Based on the results discussed in this section we find that the PM-HH method indeed finds a larger number of (effective) chains within a reasonable time, but this seems to come at the cost of the quality of

Metric	Generation 1	Generation 2
Average Reach(%)	13.65	46.37
Std Dev	0.19	0.90
Average Recall(%)	12.06	88.25
Std Dev	0.49	2.36

Table 5.6: Summary of reach (%) and recall (%) for PM-HH after one and two generations. Average and standard deviation are calculated over 20 iterations.

the ordering induced by support value. We suspect this is a consequence of the limited definition of good sub-sequences and not a limitation of the method itself. Further experiments will be necessary to determine whether this is actually the case. This would be an intersting topic for future research.

5.4.1 Experimental Setup

We test the PM-HH approach for Trapdoor parameter settings as outlined in Section 2.1.1, SA parameter settings as outlined in Section 2.2.2.3 and PM-HH parameter setting as outlined in Section 3.3.5 and Section 3.4.3. We look for chains of maximum length 4.

We test PM-HH for a sub-sequence length of 15. This value was chosen because for this value M-HH still only discovers a small percent of all effective chains, meaning that there is room for improvement in terms of reach of chains space. We only use 2 generations, so that we can evaluate the effect that a single population update can have.

In each generation 250 sequences are generated, meaning that 500 sequences are generated in total, so that we may compare the result to a mining technique tested in Experiment 3. We used a population size of 10.

To assess the consistency of our results from a single population mining iteration, we conduct 20 iterations, each involving the generation of sequences, subsequent mining for chains, population update, second sequence generation and second mining iteration.

5.4.2 Reach and Recall

Table **??** summarizes the results for the population mining method for both of its generations, in terms of percentages for reach and recall.

We find that the average reach increases from 13.65% (±0.19%) in the first generation to 46.37% (±0.90%) after a population update has occurred. This means that after the second generation, PM-HH mining has found almost of all chains possible chains it can find (50%), with the reach improving substantially.

Similarly, the average recall increases from 12.06% (±0.49%) in the first generation to 88.25% (±2.36%) in the second generation. This indicates that the PM-HH approach discovers the vast majority of effective chains by the second generation. The sharp increase in recall suggests that a population method is an effective approach to discovering more effective chains. A more detailed list of which chains are discovered by PM-HH can be found in Appendix D.

5.4.3 Time

During running we have kept track of the time necessary for the generation of sequences as well as the mining process. We have done this for both generations separately. These results are presented in Table 5.7 in seconds and in terms of chain checks. Figure 5.7 visualizes the time spent on different phases of the algorithm in seconds as well.

We observe that the time spent on both sequence generation and mining phases changes after a population update has occurred. The average time needed to generate sequences decreases, from 134.16 (±7.16) to 64.95 (±6.54) seconds, while the average time taken for mining increases, from 14.28 (±6.83) to 211.57 (±30.20) seconds.

Metric	**Generation 1**	**Generation 2**	**Total PM-HH**
Time Sequence generation			
Average (s)	134.16	64.95	199.11
Std Dev (s)	7.16	6.54	13.70
Average (cc)	7.89	3.82	11.71
Std Dev (cc)	0.42	0.38	0.81
Time Mining			
Average (s)	14.28	211.57	225.85
Std Dev (s)	6.83	30.20	37.03
Average (cc)	0.84	12.45	13.29
Std Dev (cc)	0.40	1.78	2.18
Total Time			
Average (s)	148.44	276.52	424.96
Std Dev (s)	9.48	30.74	40.22
Average (cc)	8.73	16.27	25.00
Std Dev (cc)	0.56	1.81	2.37

Table 5.7: Summary of time spent by PM-HH on different phases, for $len\_sub = 15$, $num\_gen = 2$ and $pop\_size = 10$, in seconds (s) and chain checks (cc). Averages and standard deviations are calculated over 20 iterations.

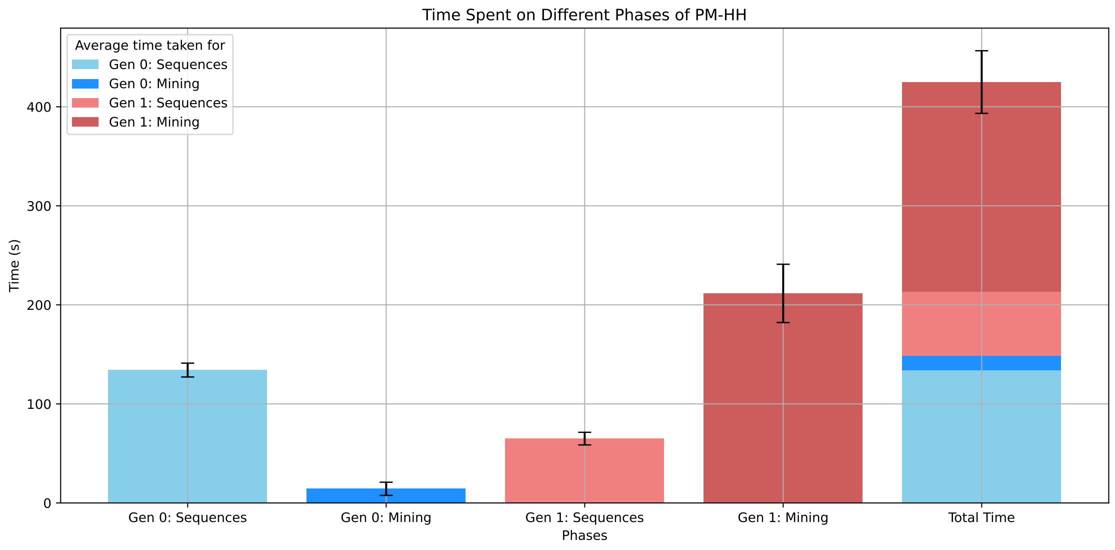


Figure 5.7: Visualization of time spent in seconds by PM-HH on different phases, averaged over 20 iterations. Standard deviations are shown in black brackets.

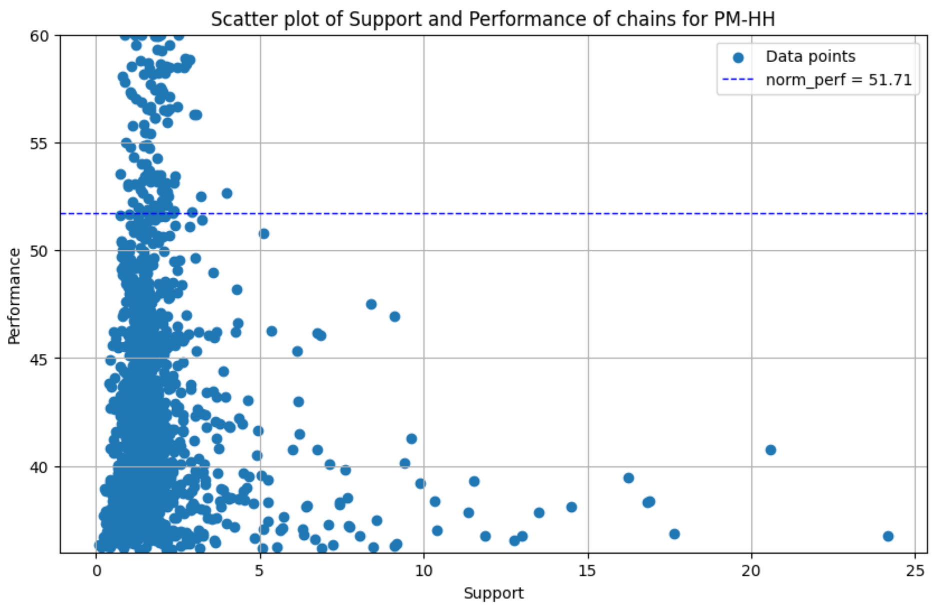


Figure 5.8: Scatter plot of support and performance of chains discovered by PM-HH for one iteration.

The fact that sequence time decreases when a population of high support operators is included aligns with the results found the preliminary experiment from Section 4.1. Here we saw that the time needed for a single SA run increases with the chain length of the global operator included, likely as a result of copying the best solution found.

The dramatic increase in mining time can be explained by the fact that in the second generation we expect many more chains to appear in each sub-sequence. Not only do the chains included as global operators appear more in a sequence, but also all variations on their sub-chains, and chains in which these sub-chains appear. For each possible relation, the chain search algorithm recurses. If more chains appear and thus more relations exist in a sub-sequence, then more recursions are necessary. This means more time will be necessary for the chain search algorithm and as a result also for the total mining step.

The average total time need for the PM-HH algorithm is 424.96 ($\pm$ 40.22) seconds, which corresponds to only 25 ($\pm$ 2.37) chains checks. This again is a relatively low number of chain checks, considering that more than a 1000 chains appear in the ordered list returned.

5.4.4 Relationship Support and Performance

Next, we look into the relationship between performance and support, as we did for the M-HH approach. Figure 5.8 shows the scatterplot for support and performance of chains discovered by PM-HH for one iteration, created as before. However, it is clear that no significant positive relation exists between support and performance in this case. In fact, we observe that many of the chains with the highest support value have relatively low performance (considerably lower than f_b).

This result is surprising, since for M-HH a significant positive relationship existed between support and performance. We suspect that this result is a natural consequence of the relatively simple definition we have used for good sub-sequences. Namely, a good subsequence is any sub-sequence with a maximum increase in objective value at the end. However, the total change of the objective value from the start of the sub-sequence to the end is not taken into account. This means that the set of good sub-sequences not only includes sub-sequence where the total objective value increases over the sub-sequence but also sub-sequences where the objective value actually is the same at the end of the sub-sequence or even has decreased. This means that large changes in objective value which only occur because we move back and forth in the solution space, are also seen as desirable behaviour, which the algorithm attempts to learn from.

We verify this by looking at the 20 chains which have the highest average position in the ordering induced by support values calculated by PM-HH. These chains can be found in Table 5.8. Notably, almost all of these

Chain	**Performance**	**Average Position**
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	40.76	8.15
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	38.52	12.80
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi$	37.86	13.10
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	36.74	14.30
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	36.74	14.70
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	36.86	14.85
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi$	37.88	15.55
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	39.46	15.55
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi$	38.20	15.63
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	42.92	17.12
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{RM} \varphi$	37.00	18.12
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	42.28	18.14
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{RM} \varphi$	36.58	21.80
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	36.26	23.15
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi$	37.04	26.65
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi$	37.48	28.40
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	40.10	28.65
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	36.32	29.10
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	36.36	32.68
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi$	39.30	32.85

Table 5.8: First 20 chains with the highest average position for PM-HH over 20 iterations.

chain start with several φ_1 operators and end with either φ or $\varphi \xrightarrow{BT} \varphi$. Clearly, these are not chains which we expect to be very good at turning a room with some 1's into a room with only 0's. However, these are chains which we would expect to be very good at turning a room with only 0's into a room with at least one 1 and then back into a room with only 0's. This seems to support our claim that the algorithm is learning back and forth moves, instead of actually improving moves.

Why then, does this issue not arise in the M-HH? This is because these back and forth moves do not occur as often in a SA search without any chains included, since SA without high support chains is worse at solving the problem than SA with high support chains. This means that higher quality solutions (rooms with only 0's) are discovered later in the search (at lower temperatures), making it less likely that back and forth moves are accepted. Namely, for a back and forth move to be accepted we first would have to accept a move which dramatically decreases the objective value. This is more likely at higher temperatures than at lower temperatures.

This problem may be partly solved for by employing a more intelligent definition of a good-sub-sequence, or by excluding chains which can be pre-determined to cause back and forth behaviour, such as $\varphi \xrightarrow{BT} \varphi$. This would be an intersting topic for future research.

5.4.5 Quality of Ordering

To review the quality of the ordering discovered, we plot the number of chains which meet some minimum performance over the number of chains checked from the ordering induced by support values. These results are shown in Figure 5.9.

This graph shows that in the first roughly 250 chain checks, very few high-performance chains are discovered. This is likely because the chains that are ranked highest are chains which cause the solution to move back and forth as discussed in the previous section. However, after around 250 chains checks we do

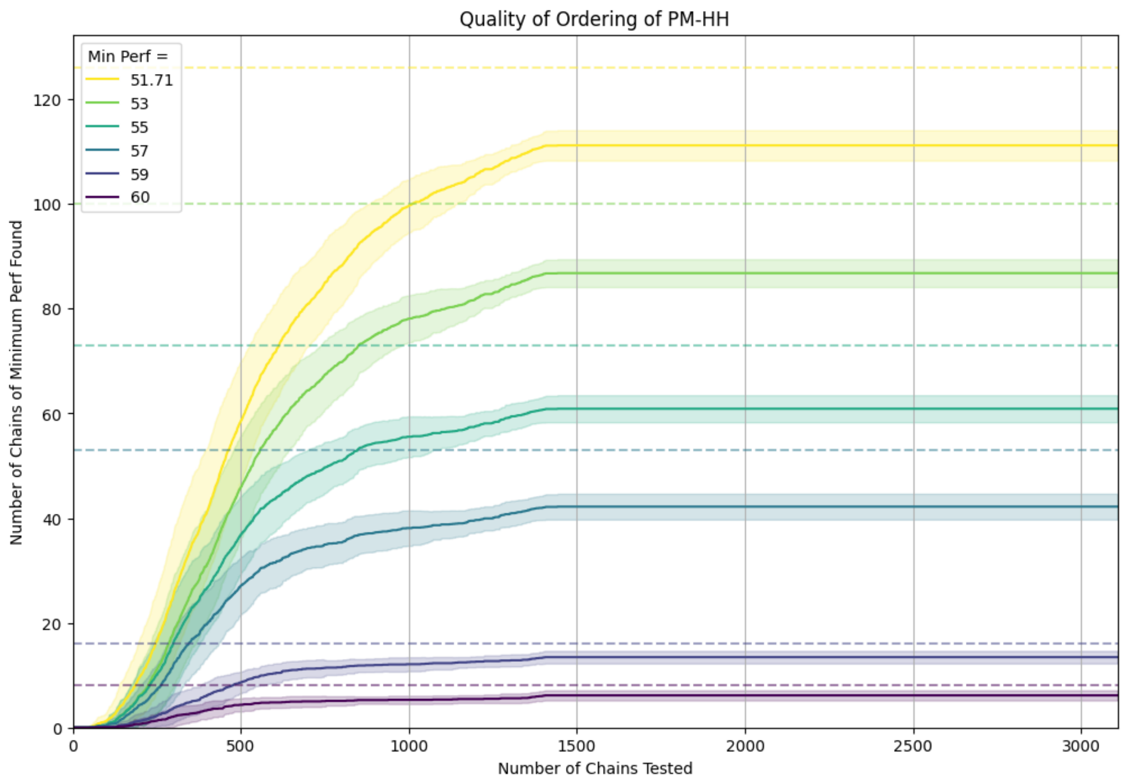


Figure 5.9: Quality of Ordering of PM-HH for different values of *min_perf* averaged over 20 iterations. The standard deviation is shown by the shaded area around the curve.

see a sharp increase in the number of high performance chains discovered. This suggest that, even though the chains are not at the top of the list, the algorithm still manages to discover high performance relatively efficiently eventually. This is the case for all minimum performance values. We also see that a little before 1500 chain checks the number of high performance chains discovered plateaus, slightly below the maximum value for each minimum performance. This makes sense, as the list of chains to check is only 1441 chains long.

5.4.6 Important Chains

Finally, we review how efficiently PM-HH finds the relevant chains to the problem. This is shown in Table 5.9. We find that all relevant chains are found relatively late, after at least 360 chain checks. In addition to this, the standard deviation for the position of all chains is relatively high as well, indicating that relevant chains are not ranked consistently at this level either.

5.5 Comparison of Hyper-Heuristics

In this section, we compare the three HHs – Priority Tree Search, Mining Hyper-Heuristic, and Population Mining Hyper-Heuristic – in terms of reach, recall, total running time, the discovery of important chains and quality of order found.

For clarity only two versions of M-HH are included in this section: $len\_sub = 15$, so we may compare this version to PM-HH for $len\_sub = 15$, and M-HH with $len\_sub = 35$, as it finds the highest number of effective chains. We will refer to these methods as M-HH (15) and M-HH (35) respectively.

Based on the results discussed in this section we find that PTS provides the highest reach of chains space and the shortest running time. However, the ordering induced by this algorithm does not effectively prioritize high-performance chains, nor does it discover the φ_0^2 chain efficiently.

Chain	Metric	PM-HH
φ_0^2	Mean Position	366.05
	Std Dev	115.41
	Found (%)	100.00
φ_0	Mean Position	490.80
	Std Dev	85.33
	Found (%)	100.00

Table 5.9: Summary of average positions of relevant chains in ordering returned by PM-HH. For each chain the average position and standard deviation are calculated over 20 iterations. The percentage of iterations for which the chain is discovered is shown as well (Found).

Metric	PTS	M-HH (15)	M-HH (35)	PM-HH
Average Reach (%)	100.00	15.61	46.41	46.37
Std Dev	-	0.55	0.53	0.90
Average Recall (%)	100.00	13.53	71.23	88.25
Std Dev	-	0.30	5.36	2.37

Table 5.10: Summary of averages and standard deviations of reach (%) and recall (%) for different HHs. Average and standard deviation are calculated over 20 iterations.

In comparison, mining-based methods take some additional time to generate an ordering of chains, but this ordering is of higher quality. M-HH (35) most effectively prioritizes high performance in its ordering. However, the method does not discover all effective chains. In contrast, PM-HH has higher recall and reach, but its ordering has many low-performance chains ranked relatively high.

In terms of which HH is most effective at discovering a high number of effective chains in a limited time, we found that different HHs are appropriate for a different number of chain checks allowed (i.e. time available). If very little time is available then M-HH (35) is most appropriate, if more time is available then PM-HH is most appropriate, and if there is almost time to check all chains then PTS is most appropriate. PM-HH and M-HH (35) also discover the most relevant chain (φ_0^2) earlier than PTS.

Based on these results we conclude that mining-based HHs such as M-HH and PM-HH are promising techniques for the generation of chains which improve the quality of an SA search. Future research is necessary to test the effect on running time of including more relations, and basis operators or allowing longer chains. However, in this research, we found that these methods already outperformed PTS for a relatively low number of relations, operators and a short chain length. We suspect that these differences will only increase for larger chain spaces.

5.5.1 Reach and Recall

Table 5.10 summarizes the number and percentages of total and effective chains found by each HH. Since PTS eventually tests all possible chains we can say that 100% of normal and effective chains appear in its ordering.

When comparing M-HH (15) and PM-HH, which are two methods which use the same sub-sequences length and generate the same number of sub-sequences, we find that PM-HH dominates M-HH both in terms of reach and recall. This suggests that a population method is an effective approach to discovering more chains without increasing the sub-sequence length. PM-HH also has roughly the same reach as M-HH (35) and even has a significantly higher recall.

5.5.2 Time

Table 5.11 shows the average time taken by each HH, with an additional row converting the total time to the equivalent number of chain checks. PTS does not have a time value here since the underlying structure

Metric	**PTS**	**M-HH (15)**	**M-HH (35)**	**PM-HH**
Average Total Time (s)	-	304.74	707.57	424.96
Std Dev (s)	-	45.07	41.49	40.22
Average Total Time (cc)	-	17.93	41.62	25.00
Std Dev (cc)	-	2.65	2.44	2.37

Table 5.11: Summary of the total time taken in both seconds (s) and chain checks (cc) for different HHs. Average and standard deviation are calculated over 20 iterations.

of a heap takes almost no additional time.

M-HH (35) takes the longest time, corresponding to approximately 41.62 (± 2.44) chain checks. Notably, PM-HH is more efficient, requiring the equivalent time of about 25.00 (± 2.37) chain checks. This means that compared to M-HH (35), PM-HH discovers more effective chains in less time. M-HH (15) is the quickest, corresponding to 17.93 (± 2.65) chain checks. However, as we saw previously, this also comes at the cost of finding only a small percentage of effective chains and not discovering the most relevant chain φ_0^2.

5.5.3 Quality of Ordering

Figure 5.10 illustrates the average percentage of effective chains found as a function of the number of chains tried for each HH. We can divide the graph into three main phases in which another method dominates:

- **Initial Discovery** (0-700 chain checks): M-HH (35) rapidly finds a large percentage of effective chains in the initial stages, outperforming both PTS, M-HH (15) and PM-HH in the early phases.
- **Intermediate Discovery** (700-1900 chains checks): PM-HH overtakes M-HH (35) in terms of the number of chains found. M-HH (35) begins to plateau as it no longer discovers high performing chains. The PM-HH methods eventually also plateaus at around 1440 chains checked.
- **Long-Term Discovery** (1900+ chain checks): After 1900 chains checks PTS eventually catches up and surpasses the other methods, as it exhaustively tests all chains. However, the rate at which it finds effective chains is slower initially compared to PM-HH and M-HH.

The shaded areas around each curve in Figure 5.10 represents the variance ($\pm std$). M-HH has a low initial variance, indicating that it consistently finds this number of chains in the initial stages. However, the variance increases as it plateaus, indicating that the total number of effective chains its finds at all may vary. PM-HH has a more consistent variance across all phases. It varies more than M-HH in the initial stages, but less in the later stages. This indicates that it is a less consistent method for cases with only very limited time, but is a more consistent method for discovering all effective chains if more time is available. No variance is shown for PTS, as only 1 run was performed. For PTS, we expect that no variance occurs, as long as enough runs are used to determine performance of a chain.

5.5.4 Important Chains

Table 5.12 summarizes the position of two most relevant chains ()φ_0^2 and φ_0) for the ordering returned by the different HH methods. A similar table, including all effective chains, can be found in Appendix D. In this table, we show the average position and standard deviation of the chain over 20 runs, as well as the percentage of runs in which the chain was discovered.

The most important chain, φ_0^2, is discovered in the least amount of chain checks by M-HH (35), namely 42.10 (± 15.68) on average. The chain is also discovered by PTS and PM-HH, but after considerably more chain checks, 777 and 366.05 (± 115.41) respectively. M-HH (15) does not discover this chain. The fact that M-HH (35) ranks this chain so consistently high are promising, as the low ranking in PTS suggest that this chain is hard to find without mining based techniques.

The second most important chain, φ_0, is discovered after 9 chain checks by PTS. M-HH (15) also finds the chain relatively early, after on average 110.70 (± 14.38) chain checks. M-HH (35) and PM-HH also find the

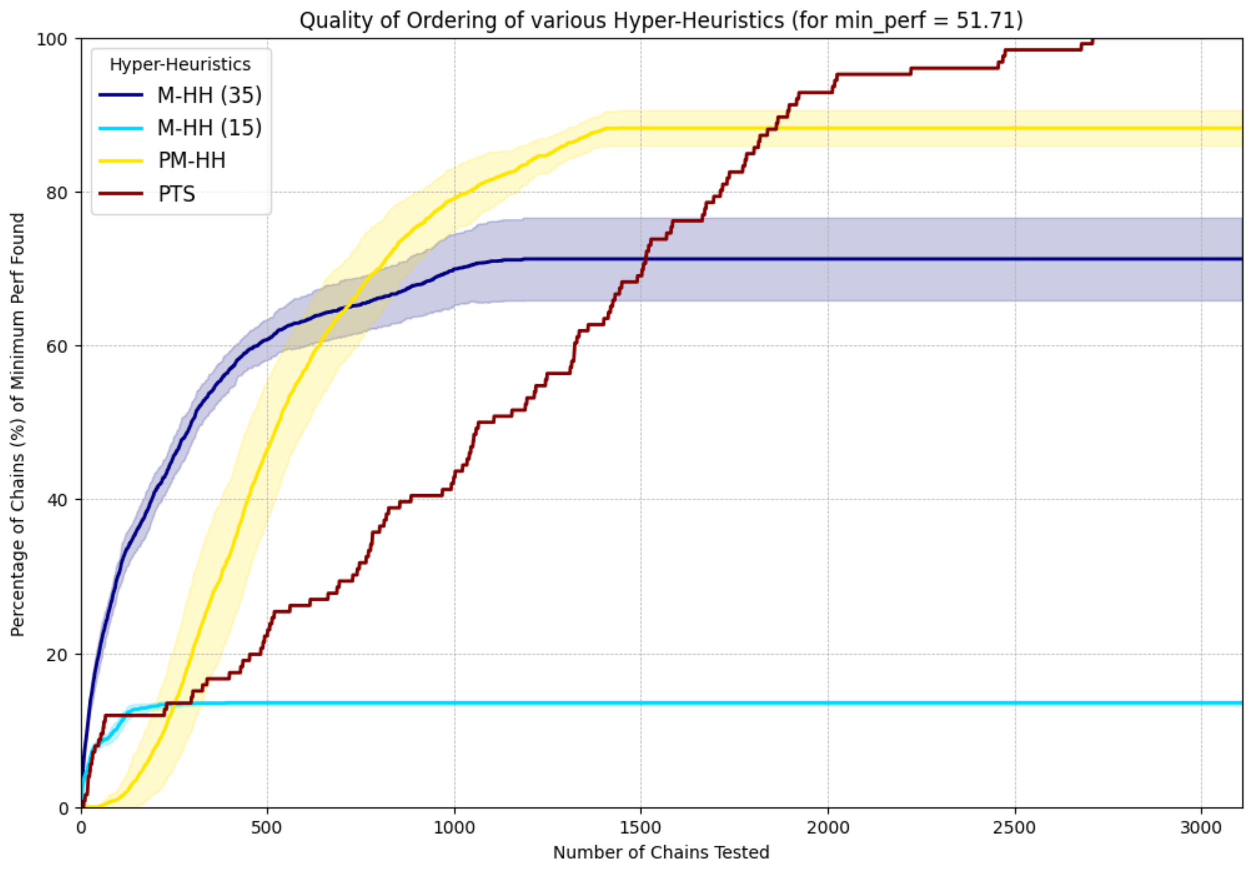


Figure 5.10: Comparison Quality of Ordering of PTS, M-HH (15), M-HH (35) and PM-HH. The standard deviation is shown by the shaded area around the curve.

Chain	**Metric Type**	**PTS**	**M-HH (15)**	**M-HH (35)**	**PM-HH**
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	Mean Position	777	-	42.10	366.05
	Std Dev	-	-	15.68	115.41
	Found (%)	-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi$	Mean Position	9	110.70	390.55	490.80
	Std Dev	-	14.38	49.43	85.33
	Found (%)	-	100.00	100.00	100.00

Table 5.12: Summary of average positions of relevant chains in ordering returned by different HHs. For each chain the average position and standard deviation are calculated over 20 iterations. The percentage of iterations for which the chain is discovered is shown as well (Found).

chain, but after considerably more chain checks, 390.55 (± 49.43) and 490.80 (± 85.33) respectively. These results suggest that for such short chains, mining-based techniques may not be a very good fit.

In summary, M-HH (35) excels in finding a longer chain like φ_0^2 more efficiently than other methods, highlighting the strength of mining-based approaches for such tasks. However, for simpler chains like φ_0, traditional methods like PTS and simpler mining techniques like M-HH (15) prove to be more efficient. This indicates a trade-off between the complexity of chains and the suitability of different HH methods for discovering them. Ultimately, finding more complex chains is highly desirable since a larger chain space is more likely to include better and more effective chains. This further underscores the potential of mining-based techniques to do so.

Chapter 6

Conclusion

In this chapter, we briefly summarize the results of this thesis, as well as its implications and limitations. We will also suggest alternatives for future research on this topic.

6.1 Summary of Findings

This research explored the development of Hyper-Heuristics (HHs) for the generation of operators to be applied in search heuristics, specifically focusing on Simulated Annealing (SA). The techniques described can be applied to general combinatorial optimization problems but were specifically tested on the Trapdoor Problem.

6.1.1 Chains

A novel element introduced here is the chain format which can be used to generate new operators by combining several operators, linked by relations which limit the locations each sub-operator can be applied to. By specifying a set of basis operators, a set of relations and setting a maximum chain length we then have an enumerable set of potential operators.

The introduction of relations allows us to limit the neighbourhood size induced by a composite operator, as well as incorporate the most basic information available on the problem. In principle, such relations can be deduced from the problem definition and do not require a user to understand how to solve the problem exactly. In any case, such relations are more easily devised than an effective operator designed from scratch.

The performance of a chain was tested by running SA 50 times for the 6x6 Trapdoor Problem, with the bitflip operator and the chain as global operators. We tested the performance of chains of maximum length 4, constructed from 2 basis operators and 3 relations. Of these 3108 chains, 126 chains were found to significantly improve the average objective value of the solution found by SA over 50 runs, as compared to SA with only the bitflip operator as a global operator.

6.1.2 Hyper-Heuristics

In addition to this, using the chain format three HHs were developed, each of which returns an ordered list of chains to test. The objective of these HHs is to effectively prioritize high-performing chains, ensuring that even with limited computational time, high-performance chains are still discovered.

Priority Tree Search (PTS) was designed to prioritize chains based on the performance of their subchains, relying solely on the average objective value of a solution after 50 SA runs. Although PTS eventually managed to test all chains, it was less efficient in discovering high-performing chains early due to its assumption that subchains of good chains are also good chains, which does not hold true for the Trapdoor Problem, and thus also does not hold true generally.

The Mining Hyper-Heuristic (M-HH) analyzes the effect of different chains as they appear within an SA search, leveraging more of the information contained in a single run of SA. Based on the frequencies of chains in sections of the search where desirable behaviour is observed, a support value is calculated. This support value is used to create an ordering of chains. A significant positive relationship was discovered between support and the performance of a chain, suggesting that these support values are good predictors of a chain's effectiveness within SA. However, support values could not be calculated for several chains due to the rarity of their appearance within the limited sections of the search.

The Population Mining Hyper-Heuristic (PM-HH) method aims to address some of the shortcomings of M-HH with a population system, designed to ensure that chains appear more frequently in short sections of search. Compared to M-HH with similar parameters, PM-HH effectively increased the number of discovered chains. However, this seems to come at the cost of the quality of the ordering induced by support value. We find that this is a consequence of the relatively basic definition of good sub-sequences and not a limitation of the method itself.

6.2 Implications of the Research

This research demonstrates the potential of chain structures and mining-based techniques in improving the quality of search heuristics. The findings from testing these HHs on the Trapdoor Problem suggest that these techniques can be useful for larger and more complex problems, warranting further exploration.

The discovery of several chains that significantly improved the quality of the search implies that the chain structure effectively generalizes the components necessary to create new operators through the combinations of simpler operators. Such a technique for generating operators from basic components may be useful not only in the HHs explored in this thesis, but also in alternative HH approaches such as evolutionary computing, or grammatical evolution. Such a structure allows the user to incorporate the most basic knowledge available of the problem to generate new chains. While this might imply that the HHs are not entirely domain-independent, it only requires a superficial understanding of the problem to determine some relevant relations, making it accessible and broadly applicable.

The findings from PTS suggest that although some chains do exist which improve performance, they can be hard to find, especially as the complexity of chains increases. This highlights the need for time-efficient HHs, which are not as sensitive to the complexity of chains.

The significant positive relation discovered between support and performance of chains within M-HH demonstrates that the information contained in a single run of SA is useful for the identification of effective chains for the Trapdoor Problem. This further implies that such a technique may also be effective for other, more complex, optimization problems, which also struggle with escaping from local optima. This is further supported by the fact that M-HH outperforms PTS in terms of the quality of ordering for limited computational time, and consistently finds the most important chain (φ_0^2) within the least amount chain checks out of all discussed HHs.

The results obtained for PM-HH indicate that M-HH can be further enhanced by incorporating a population-based system. This approach enables the algorithm to discover longer chains while keeping the length of subsequences short, thereby reducing the computational time required to create an ordering. PM-HH proves to be more suitable for identifying longer chains, as it is less constrained by the natural frequency of chains within the search process. Efficiently discovering longer chains increases the likelihood of finding higher-performing chains, as it expands the search space. Although PM-HH experiments revealed the discovery of many low-performance chains that caused the solution to oscillate, this is likely due to the limited definition of desirable behaviour rather than an inherent flaw in the technique. Overall, PM-HH shows promise in uncovering longer, more effective chains, provided the criteria for desirable behaviour are refined.

6.3 Limitations and Future Research

This research has certain limitations that present opportunities for further investigation. This section outlines these limitations and suggests directions for future research to address them.

The most notable limitation of this research is the definition of good sub-sequences. As discussed in Section 5.4, refining this definition could significantly enhance the effectiveness of mining-based HHs. Currently, the simplistic definition often leads to the discovery of ineffective back-and-forth chains. A more nuanced definition could consider the objective value at the start and end of a sub-sequence. Alternatively, one could directly take into account the objective value before and after a chain appears in a sub-sequence by examining the objective value associated with its first and last sub-operators. This approach would allow us to directly dismiss ineffective chains based on information already contained within the sub-sequences.

In addition to this limitation, several others exist. In this section, we discuss these limitations and suggest several options for potential future research to address them: the further development of HHs discussed in this thesis, more extensive testing of HHs, and research into the combination of HHs.

6.3.1 Further Optimizing Hyper-Heuristics

There are several other ways in which the HHs discussed in this thesis may be improved.

We expect that the PTS algorithm could be made significantly more time efficient by dismissing chains with relatively low performance after fewer SA runs. For instance, an additional step could be introduced to prioritize which leaves to test further based on preliminary performance. This way, relatively less time is wasted on low-potential chains. However, despite these improvements, PTS may still lag behind mining-based approaches due to its reliance on a false assumption and the fact that similar tactics could benefit mining-based heuristics. For this reason, we suggest future research focuses more on the further development of the mining-based heuristics discussed in this thesis.

In terms of the mining-based heuristics, it may be useful to employ an even more comprehensive definition of good sub-sequences i.e. by using other information available during the search. For example, only considering sequences in which the objective value reaches a new maximum, or taking into account the number of iterations since the last improvement in objective value. One could even use a combination of definitions to find a set of good-sub-sequences. Extensive research into the effects of different definitions of good sub-sequences and desirable behaviour could further enhance the effectiveness of mining-based techniques.

The HH methods discussed in this thesis still require the tuning of several parameters. Although the study aimed to minimize this necessity, some level of parameter tuning remains essential for optimal performance. This can be time-consuming and may result in replacing the tuning of one set of parameters (which operators to include) for another (which parameters to use in the HH). Regarding parameter settings, further research is necessary to determine the impact of various parameters in greater detail. As more insights become available on the effects of parameters in M-HH and PM-HH, it may become easier to decide the most effective parameters based on the available information about the problem and the heuristic implementation. This would reduce the need for parameter tuning, thereby making the technique more generalizable across different domains.

Finally, a potentially effective approach which was not explored in this thesis is to directly implement discovered chains as basis operators from which to build new chains. This would require relations between chains to be defined, potentially based on a set of basis-relations between basis operators as discussed in this research. By using chains directly as basis operators we create an even larger chain space, with the same amount of initial relations and basis operators, meaning that even more effective operators may be defined and discovered in this way. Initial research could apply the HHs in a multi-phase process: first, discover high-performance chains, and then use these chains as basis operators in subsequent HH applications.

6.3.2 A More Complete Test

Although the mining-based techniques developed in this thesis show potential for discovering effective operators in a limited time, they have only been tested in a relatively simple context: for the Trapdoor Problem, with a simple implementation of Simulated Annealing and for a limited chain space.

Efforts have been made to ensure this context is sufficiently challenging. For instance, the Trapdoor Problem, despite its simplicity, includes local optima that can be difficult to escape, providing a meaningful challenge for evaluating the techniques. In terms of SA, the implementation includes incremental objective value evaluation and an exponential decay schedule to mimic more realistic scenarios. Additionally, the chain space, though limited to two basis operators and three relations with a maximum chain length of four, results in 3108 possible chains to test. This provides a substantial, though manageable, search space for initial exploration.

While this context is suitable for a proof-of-concept approach, i.e. demonstrating the feasibility and potential of the techniques, the promising results may not generalize to larger or different types of problems without further validation.

To this end, future research should focus on testing the mining-based techniques on a variety of problems from different domains, as well as alternative implementations of SA. For example, as the HHs were originally designed to more efficiently solve a variety of Vehicle Routing problems using SA, this may be a good place to start. It may also be interesting to test the effectiveness of the developed chain structure and HHs on problems where constraints are incorporated into the objective function. We expect that such problems have a search space with many local extremes which are hard to escape without changing a large part of the solution. Operators which consist of several sub-operators, may more effectively perform the correct sequence of steps needed to escape the optimum.

To further determine the added benefit of the HH, future research could focus on testing the HHs ability to discover operators for SA (or other search heuristics) for which considerable effort has already been put into the optimization of parameters and operators included. An example of such an implementation is CQM's RitOpt 4 algorithm, as discussed in Section 1.3.1. For such implementations, we expect it will be harder to discover new operators which improve performance. In a similar vein, the developed HHs will need to be compared to other HHs aimed at the generation of perturbative operators, such as the evolutionary computing techniques discussed in Chapter 1.2.

In particular, it is important to evaluate the sensitivity of these techniques to the size of the allowed chain space, induced by the number of relations, basis operators and maximum chain length. A larger chain space likely contains more effective operators but is also more time-consuming to explore. A simple, yet effective way of testing this would be to introduce dummy relations, which randomly return that a relation holds true with a fixed probability. Effective HHs should rank chains with this relation relatively low.

6.3.3 Combination of Techniques

Finally, as we saw in the experiments discussed in 5, different HHs exhibit different desirable properties. For example, PTS does not induce the most effective ordering, but it does discover all chains. In contrast, M-HH returns the most effective ordering of chains but does not discover many chains which would improve performance.

These results suggest that to design a HH approach which performs well across all Key Performance Indicators, it may be useful to research HH approaches which combine characteristics of both PTS and the mining-based approaches. For example, an adaptation of PTS could determine which leaf in the tree to expand and test based on support values found by M-HH or PM-HH. Alternatively, after all chains returned by M-HH or PM-HH have been tested, the remaining chains which were not discovered could be tested using PTS.

Future research may also look into the integration of techniques and structures described in this thesis with alternative HH approaches. For example, evolutionary computing techniques may also be applied in combination with a chain-based grammar.

6.4 Conclusion

In conclusion, this thesis has developed and tested several HH techniques for generating operators in Simulated Annealing, particularly focusing on the Trapdoor Problem. The novel introduction of chain structures has shown significant promise in enhancing search heuristics by combining simpler operators into more effective composite operators. The development of Priority Tree Search, Mining Hyper-Heuristic, and Population Mining Hyper-Heuristic has provided valuable insights into the strengths and limitations of different HH approaches.

While the findings are promising, several limitations need to be addressed, including the need for more extensive testing on diverse and complex problems, refinement of parameter settings, and improving the definition of desirable behaviour. Future research should focus on improving these HHs, testing their generalizability, and exploring combinations with other heuristic approaches.

Overall, the techniques developed in this thesis offer a solid foundation for further exploration and development of HHs, with the potential to significantly improve the efficiency and effectiveness of search heuristics in solving complex optimization problems.

Bibliography

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. Santiago. 1994, pp. 487–499.

[2] David L. Applegate et al. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

[3] Mohamed Bader-El-Den and Riccardo Poli. "Generating SAT local-search heuristics using a GP hyper-heuristic framework". In: *International conference on artificial evolution (evolution artificielle)*. Springer. 2007, pp. 37–49.

[4] Edmund K Burke, Matthew R Hyde, and Graham Kendall. "Grammatical evolution of local search heuristics". In: *IEEE Transactions on Evolutionary Computation* 16.3 (2011), pp. 406–417.

[5] Edmund K Burke et al. "A classification of hyper-heuristic approaches". In: *Handbook of metaheuristics* (2010), pp. 449–468.

[6] Edmund K Burke et al. "Hyper-heuristics: A survey of the state of the art". In: *Journal of the Operational Research Society* 64 (2013), pp. 1695–1724.

[7] Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. "Automatic design of hyper-heuristic based on reinforcement learning". In: *Information Sciences* 436 (2018), pp. 89–107.

[8] Jesse Clifton and Eric Laber. "Q-learning: Theory and applications". In: *Annual Review of Statistics and Its Application* 7.1 (2020), pp. 279–301.

[9] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Logic, automata, and computational complexity: The works of Stephen A. Cook*. ACM, 2023, pp. 143–152.

[10] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[11] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1998.

[12] Mayur Datar et al. "Maintaining stream statistics over sliding windows". In: *SIAM journal on computing* 31.6 (2002), pp. 1794–1813.

[13] Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. "Simulated annealing: From basics to applications". In: *Handbook of metaheuristics* (2019), pp. 1–35.

[14] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. "Reinforcement learning with combinatorial actions: An application to vehicle routing". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 609–620.

[15] Jay L Devore. "Probability and statistics". In: *Pacific Grove: Brooks/Cole* (2000).

[16] Luca Di Gaspero and Andrea Schaerf. "Neighborhood portfolio approach for local search applied to timetabling problems". In: *Journal of Mathematical Modelling and Algorithms* 5 (2006), pp. 65–89.

[17] Luca Di Gaspero and Tommaso Urli. "Evaluation of a family of reinforcement learning cross-domain optimization heuristics". In: *International conference on learning and intelligent optimization*. Springer. 2012, pp. 384–389.

[18] Kathryn A. Dowsland and Jonathan M. Thompson. "Simulated Annealing". In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Springer, Berlin, Heidelberg, 2012, pp. 1623–1655.

[19] Matthias Englert, Heiko Röglin, and Berthold Vöcking. “Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP”. In: *Algorithmica* 68.1 (2014), pp. 190–264.

[20] Thomas A Feo and Mauricio GC Resende. “Greedy randomized adaptive search procedures”. In: *Journal of global optimization* 6 (1995), pp. 109–133.

[21] Alex S Fukunaga. “Automated discovery of local search heuristics for satisfiability testing”. In: *Evolutionary computation* 16.1 (2008), pp. 31–61.

[22] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[23] Jun Gu and Xiaofei Huang. “Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP)”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 24.5 (1994), pp. 728–735.

[24] Keld Helsgaun. “An effective implementation of the Lin–Kernighan traveling salesman heuristic”. In: *European journal of operational research* 126.1 (2000), pp. 106–130.

[25] Karla L Hoffman. “Combinatorial optimization: Current successes and directions for the future”. In: *Journal of computational and applied mathematics* 124.1-2 (2000), pp. 341–360.

[26] John Hopcroft and Robert Tarjan. “Algorithm 447: efficient algorithms for graph manipulation”. In: *Communications of the ACM* 16.6 (1973), pp. 372–378.

[27] David S Johnson. “Local optimization and the traveling salesman problem”. In: *International colloquium on automata, languages, and programming*. Springer. 1990, pp. 446–461.

[28] Maryam Karimi-Mamaghan et al. “A learning-based iterated local search algorithm for solving the traveling salesman problem”. In: *International conference on optimization and learning*. Springer. 2021, pp. 45–61.

[29] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. “A review on genetic algorithm: past, present, and future”. In: *Multimedia tools and applications* 80 (2021), pp. 8091–8126.

[30] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. “Optimization by simulated annealing”. In: *science* 220.4598 (1983), pp. 671–680.

[31] Natallia Kokash. “An introduction to heuristic algorithms”. In: *Department of Informatics and Telecommunications* (2005), pp. 1–8.

[32] Bernhard H Korte et al. *Combinatorial optimization*. Vol. 1. Springer, 2011.

[33] John R Koza. “Genetic programming as a means for programming computers by natural selection”. In: *Statistics and computing* 4 (1994), pp. 87–112.

[34] Alexandre Laterre et al. “Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization”. In: *arXiv preprint arXiv:1807.01672* (2018).

[35] Joseph YT Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. Chapman and Hall/CRC, 2004.

[36] Shen Lin and Brian W Kernighan. “An effective heuristic algorithm for the traveling-salesman problem”. In: *Operations research* 21.2 (1973), pp. 498–516.

[37] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. “Iterated local search”. In: *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[38] Zhipeng Lü, Jin-Kao Hao, and Fred Glover. “Neighborhood analysis: a case study on curriculum-based course timetabling”. In: *Journal of Heuristics* 17 (2011), pp. 97–118.

[39] David McAllester, Bart Selman, Henry Kautz, et al. “Evidence for invariants in local search”. In: *AAAI/IAAI*. Rhode Island, USA. 1997, pp. 321–326.

[40] George Mweshi and Nelishia Pillay. “An improved grammatical evolution approach for generating perturbative heuristics to solve combinatorial optimization problems”. In: *Expert Systems with Applications* 165 (2021), p. 113853.

[41] Nelishia Pillay and Rong Qu. *Hyper-heuristics: theory and applications*. Springer, 2018.

[42] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, 2016.

[43] Jette Randlov. "Learning macro-actions in reinforcement learning". In: *Advances in Neural Information Processing Systems* 11 (1998).

[44] Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications.* Vol. 840. Springer, 2003.

[45] Stefan Ropke and David Pisinger. "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows". In: *Transportation science* 40.4 (2006), pp. 455–472.

[46] Conor Ryan, John James Collins, and Michael O Neill. "Grammatical evolution: Evolving programs for an arbitrary language". In: *Genetic Programming: First European Workshop, EuroGP'98 Paris, France, April 14–15, 1998 Proceedings 1.* Springer. 1998, pp. 83–96.

[47] Nasser R Sabar et al. "Grammatical evolution hyper-heuristic for combinatorial optimization problems". In: *IEEE Transactions on Evolutionary Computation* 17.6 (2013), pp. 840–861.

[48] Melissa Sánchez et al. "A systematic review of hyper-heuristics on combinatorial optimization problems". In: *IEEE Access* 8 (2020), pp. 128068–128095.

[49] Bart Selman, Henry Kautz, et al. "Domain-independent extensions to GSAT: Solving large structured satisfiability problems". In: *IJCAI.* Vol. 93. 1993, pp. 290–295.

[50] Bart Selman, Henry A Kautz, Bram Cohen, et al. "Noise strategies for improving local search". In: *AAAI.* Vol. 94. 1994, pp. 337–343.

[51] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[52] Finnegan Southey. "Constraint metrics for local search". In: *Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005, St Andrews, UK, June 19-23, 2005. Proceedings 8.* Springer. 2005, pp. 269–281.

[53] Frank R Spellman and Nancy E Whiting. *Handbook of Mathematics and Statistics for the Environment.* CRC Press, 2013.

[54] Christopher Stone, Emma Hart, and Ben Paechter. "On the synthesis of perturbative heuristics for multiple combinatorial optimisation domains". In: *Parallel Problem Solving from Nature–PPSN XV: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part I 15.* Springer. 2018, pp. 170–182.

[55] Balram Suman. "Study of simulated annealing based algorithms for multiobjective optimization of a constrained problem". In: *Computers & chemical engineering* 28.9 (2004), pp. 1849–1871.

[56] Robert Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[57] Paolo Toth and Daniele Vigo, eds. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2002.

[58] Helmut F Van Emden. *Statistics for terrified biologists.* John Wiley & Sons, 2019.

A Estimating the Basis Performance

Let $\overline{f_c}$ be the performance measured for a chain c over 50 runs. We consider a chain c to improve performance if the corresponding 95% Confidence Interval (CI) for $\overline{f_c} - \overline{f_o}$ is strictly positive. This is the case when

$$\overline{f_c} - \overline{f_o} - t_{\alpha/2} \cdot SE > 0$$

Here $\overline{f_0} = 50.02$ as measured in Section 4.1, SE is the standard error of the difference and $t_{\alpha/2}$ the critical value. To determine what value $\overline{f_c}$ should at least have, we first need to estimate these two values.

The standard error of the difference of two averages is calculated as follows:

$$SE = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

where s_1 and s_2 are the standard deviations of the two samples, and n_1 and n_2 are the sample sizes. By assuming the standard deviation of both samples is at most s_{max}, the standard error of the difference can estimated as follows:

$$SE = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \approx \sqrt{\frac{s_{max}^2}{100} + \frac{s_{max}^2}{50}} \approx 0.85.$$

To determine the degrees of freedom for our t-distribution, we use the following formula for unequal variances (Welch-Satterthwaite equation [53, 58]):

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2 - 1}}$$

In calculating this value we assume that both standard deviations are roughly equal to $s_1 = s_2 = s_{max} = 4.93$. We can then approximate the degrees of freedom as follows:

$$df \approx \frac{\left(\frac{4.93^2}{100} + \frac{4.93^2}{50}\right)^2}{\frac{\left(\frac{4.93^2}{100}\right)^2}{99} + \frac{\left(\frac{4.92}{50}\right)^2}{49}} \approx 97.24$$

For a 95% confidence level with degrees of freedom approximating sample size 100, the critical value $t_{\alpha/2} = 1.984$ based on the t-distribution table. Whether the CI is strictly positive is then determined entirely by the measured performance of the chain. The interval is strictly positive if:

$$\overline{f_c} > \overline{f_o} + t_{\alpha/2} \cdot SE \approx 50.02 + 1.984 \cdot 0.85 = 51.71.$$

B Regression Results M-HH

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	40.45	1.64	0.00	0.01	0.35
1	40.62	1.50	0.00	0.01	0.34
2	40.09	1.78	0.00	0.00	0.38
3	40.08	1.82	0.00	0.01	0.34
4	40.59	1.57	0.00	0.01	0.33
5	40.85	1.40	0.00	0.01	0.34
6	40.28	1.82	0.00	0.01	0.35
7	39.81	1.87	0.00	0.00	0.38
8	40.01	1.81	0.00	0.00	0.35
9	39.63	1.93	0.00	0.00	0.39
10	40.17	1.59	0.00	0.00	0.36
11	39.91	1.86	0.00	0.00	0.40
12	39.25	2.23	0.00	0.00	0.42
13	39.86	1.74	0.00	0.00	0.39
14	40.70	1.58	0.00	0.01	0.32
15	40.24	1.68	0.00	0.00	0.38
16	39.12	2.09	0.00	0.00	0.42
17	39.19	2.09	0.00	0.00	0.41
18	39.89	1.89	0.00	0.01	0.34
19	40.23	1.71	0.00	0.00	0.38
Average	40.05	1.78	0.00	0.00	0.37

Table 1: Regression results for len_sub = 5

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	38.67	1.79	0.00	0.00	0.44
1	38.51	1.89	0.00	0.00	0.46
2	38.47	1.92	0.00	0.00	0.43
3	38.99	1.73	0.00	0.00	0.38
4	39.14	1.65	0.00	0.00	0.40
5	38.27	1.92	0.00	0.00	0.47
6	38.54	1.95	0.00	0.00	0.45
7	38.63	1.72	0.00	0.00	0.43
8	38.33	1.86	0.00	0.00	0.47
9	38.23	2.03	0.00	0.00	0.47
10	39.24	1.39	0.00	0.00	0.42
11	37.79	1.93	0.00	0.00	0.51
12	38.69	1.83	0.00	0.00	0.44
13	38.78	1.60	0.00	0.00	0.45
14	38.93	1.65	0.00	0.00	0.44
15	38.50	1.76	0.00	0.00	0.46
16	38.09	2.04	0.00	0.00	0.51
17	37.65	2.23	0.00	0.00	0.53
18	38.16	1.95	0.00	0.00	0.44
19	39.16	1.56	0.00	0.00	0.45
Average	38.54	1.82	0.00	0.00	0.45

Table 2: Regression results for len_sub = 10

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	38.10	1.60	0.00	0.00	0.43
1	37.96	1.64	0.00	0.00	0.45
2	38.50	1.53	0.00	0.00	0.35
3	38.50	1.51	0.00	0.00	0.32
4	38.63	1.37	0.00	0.00	0.38
5	37.87	1.65	0.00	0.00	0.42
6	38.06	1.79	0.00	0.00	0.39
7	38.22	1.45	0.00	0.00	0.42
8	38.15	1.56	0.00	0.00	0.42
9	37.90	1.72	0.00	0.00	0.42
10	38.48	1.22	0.00	0.00	0.43
11	37.71	1.71	0.00	0.00	0.48
12	38.17	1.61	0.00	0.00	0.42
13	38.41	1.35	0.00	0.00	0.42
14	38.18	1.60	0.00	0.00	0.43
15	37.91	1.56	0.00	0.00	0.45
16	37.37	1.89	0.00	0.00	0.50
17	37.68	1.76	0.00	0.00	0.43
18	37.74	1.71	0.00	0.00	0.43
19	38.08	1.48	0.00	0.00	0.47
Average	38.08	1.59	0.00	0.00	0.42

Table 3: Regression results for len_sub = 15

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	37.96	1.72	0.00	0.00	0.45
1	37.92	1.72	0.00	0.00	0.45
2	37.88	1.90	0.00	0.00	0.43
3	38.06	1.76	0.00	0.00	0.36
4	38.37	1.62	0.00	0.00	0.38
5	37.98	1.65	0.00	0.00	0.42
6	37.94	1.86	0.00	0.00	0.42
7	38.54	1.36	0.00	0.00	0.42
8	38.08	1.59	0.00	0.00	0.45
9	37.77	1.74	0.00	0.00	0.49
10	38.00	1.44	0.00	0.00	0.48
11	37.84	1.73	0.00	0.00	0.49
12	38.08	1.79	0.00	0.00	0.44
13	38.21	1.54	0.00	0.00	0.45
14	38.04	1.76	0.00	0.00	0.45
15	37.75	1.68	0.00	0.00	0.48
16	37.47	1.84	0.00	0.00	0.53
17	37.60	1.81	0.00	0.00	0.48
18	37.78	1.72	0.00	0.00	0.46
19	37.95	1.54	0.00	0.00	0.50
Average	37.96	1.69	0.00	0.00	0.45

Table 4: Regression results for len_sub = 20

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	37.55	1.98	0.00	0.00	0.48
1	37.53	1.95	0.00	0.00	0.46
2	37.64	2.18	0.00	0.00	0.45
3	37.58	2.07	0.00	0.00	0.41
4	38.01	1.98	0.00	0.00	0.41
5	37.77	1.85	0.00	0.00	0.43
6	37.72	2.10	0.00	0.00	0.44
7	38.10	1.62	0.00	0.00	0.43
8	37.48	1.91	0.00	0.00	0.46
9	37.29	1.99	0.00	0.00	0.53
10	37.36	1.78	0.00	0.00	0.49
11	37.28	2.01	0.00	0.00	0.53
12	37.92	2.07	0.00	0.00	0.45
13	37.80	1.79	0.00	0.00	0.49
14	37.73	2.09	0.00	0.00	0.45
15	37.47	1.92	0.00	0.00	0.49
16	37.30	2.01	0.00	0.00	0.54
17	37.34	2.05	0.00	0.00	0.49
18	37.42	1.98	0.00	0.00	0.45
19	37.53	1.76	0.00	0.00	0.50
Average	37.59	1.95	0.00	0.00	0.47

Table 5: Regression results for len_sub = 25

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	37.28	2.09	0.00	0.00	0.51
1	37.06	2.13	0.00	0.00	0.50
2	37.10	2.37	0.00	0.00	0.46
3	37.28	2.16	0.00	0.00	0.44
4	37.60	2.11	0.00	0.00	0.41
5	37.17	2.08	0.00	0.00	0.48
6	37.31	2.22	0.00	0.00	0.46
7	37.49	1.91	0.00	0.00	0.50
8	37.17	2.04	0.00	0.00	0.48
9	37.08	2.12	0.00	0.00	0.52
10	36.91	2.00	0.00	0.00	0.55
11	37.03	2.15	0.00	0.00	0.55
12	37.56	2.22	0.00	0.00	0.47
13	37.52	1.94	0.00	0.00	0.49
14	37.44	2.24	0.00	0.00	0.45
15	37.03	2.13	0.00	0.00	0.52
16	37.01	2.19	0.00	0.00	0.54
17	36.96	2.16	0.00	0.00	0.52
18	36.95	2.13	0.00	0.00	0.49
19	37.21	1.91	0.00	0.00	0.51
Average	37.21	2.11	0.00	0.00	0.49

Table 6: Regression results for len_sub = 30

Iteration	Intercept	Slope	P-value Intercept	P-value Slope	R-squared
0	36.89	2.16	0.00	0.00	0.47
1	37.36	2.14	0.00	0.00	0.40
2	37.36	2.43	0.00	0.00	0.38
3	36.27	2.31	0.00	0.00	0.55
4	36.89	2.10	0.00	0.00	0.48
5	36.77	2.46	0.00	0.00	0.48
6	36.42	2.31	0.00	0.00	0.52
7	36.80	2.26	0.00	0.00	0.50
8	36.84	2.13	0.00	0.00	0.49
9	36.84	2.51	0.00	0.00	0.43
10	36.76	2.58	0.00	0.00	0.45
11	36.94	2.21	0.00	0.00	0.46
12	37.04	2.20	0.00	0.00	0.44
13	36.97	2.11	0.00	0.00	0.51
14	36.97	2.15	0.00	0.00	0.46
15	36.85	2.27	0.00	0.00	0.49
16	36.98	2.09	0.00	0.00	0.51
17	37.06	2.15	0.00	0.00	0.45
18	37.17	1.90	0.00	0.00	0.48
19	36.92	2.33	0.00	0.00	0.49
Average	36.91	2.24	0.00	0.00	0.47

Table 7: Regression results for len_sub = 35

C Analysis Position Effective chains M-HH

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	-	-	-	554.83
	SD		-	-	-	-	-	-	-	287.09
	F(%)		-	-	-	-	-	-	-	30.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	-	-	-	88.25
	SD		-	-	-	-	-	-	-	56.61
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	-	-	-	209.50
	SD		-	-	-	-	-	-	-	67.81
	F(%)		-	-	-	-	-	-	-	30.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	23.16	28.15	39.10	42.10
	SD		-	-	-	-	13.00	17.11	20.18	15.68
	F(%)		-	-	-	-	95.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	-	-	-	149.00
	SD		-	-	-	-	-	-	-	96.42
	F(%)		-	-	-	-	-	-	-	30.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	245.44	275.70	287.35	285.50
	SD		-	-	-	-	77.59	88.93	82.67	69.63
	F(%)		-	-	-	-	80.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	-	-	-	600.40
	SD		-	-	-	-	-	-	-	97.40
	F(%)		-	-	-	-	-	-	-	25.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	-	-	-	-	-	-	-	68.71
	SD		-	-	-	-	-	-	-	39.30
	F(%)		-	-	-	-	-	-	-	35.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	-	-	-	-	-	5.36	6.05	4.55
	SD		-	-	-	-	-	3.02	5.97	4.47

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	F(%)		-	-	-	-	-	55.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	-	-	-	-	-	-	-	549.33
	SD		-	-	-	-	-	-	-	175.06
	F(%)		-	-	-	-	-	-	-	15.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	-	-	-	-	-	-	-	618.25
	SD		-	-	-	-	-	-	-	198.68
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	-	5.15	22.30	110.70	255.20	327.60	375.80	390.55
	SD		-	0.36	2.30	12.95	35.50	37.13	50.66	49.43
	F(%)		-	100.00	100.00	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.52	-	-	-	-	-	6.73	7.95	7.20
	SD		-	-	-	-	-	5.14	4.98	5.97
	F(%)		-	-	-	-	-	75.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.52	-	-	-	117.15	282.05	355.85	406.65	422.85
	SD		-	-	-	50.35	73.06	74.16	81.76	70.59
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	59.32	-	-	-	-	659.89	871.35	947.30	973.70
	SD		-	-	-	-	86.16	65.77	47.42	49.74
	F(%)		-	-	-	-	95.00	100.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.28	-	-	-	128.47	246.00	308.45	350.10	348.25
	SD		-	-	-	34.94	64.99	61.07	45.26	40.53
	F(%)		-	-	-	95.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.90	-	-	-	-	-	-	651.00	447.53
	SD		-	-	-	-	-	-	154.28	193.71
	F(%)		-	-	-	-	-	-	20.00	85.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.86	-	-	-	-	214.56	220.25	228.20	217.65
	SD		-	-	-	-	67.17	77.15	63.16	61.30
	F(%)		-	-	-	-	90.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.80	-	-	-	-	-	-	712.40	399.28

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	SD		-	-	-	-	-	-	231.44	189.47
	F(%)		-	-	-	-	-	-	25.00	90.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.78	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	58.68	-	-	-	-	-	-	737.00	559.47
	SD		-	-	-	-	-	-	0.00	255.89
	F(%)		-	-	-	-	-	-	5.00	85.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	58.62	-	-	-	-	406.61	462.70	506.85	528.90
	SD		-	-	-	-	152.76	150.44	153.89	126.12
	F(%)		-	-	-	-	90.00	100.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	58.62	-	-	-	-	-	-	-	751.40
	SD		-	-	-	-	-	-	-	237.40
	F(%)		-	-	-	-	-	-	-	25.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	58.60	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.58	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.54	-	-	-	-	-	-	598.57	421.50
	SD		-	-	-	-	-	-	133.42	162.29
	F(%)		-	-	-	-	-	-	35.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.52	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.50	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	58.50	-	-	-	-	-	-	-	1040.33
	SD		-	-	-	-	-	-	-	75.83
	F(%)		-	-	-	-	-	-	-	15.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	58.46	-	-	-	-	-	-	-	1104.17
	SD		-	-	-	-	-	-	-	75.62
	F(%)		-	-	-	-	-	-	-	30.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.46	-	-	-	-	-	707.00	634.62	578.05
	SD		-	-	-	-	-	0.00	241.87	224.08
	F(%)		-	-	-	-	-	5.00	65.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.42	-	-	-	-	-	-	-	566.00
	SD		-	-	-	-	-	-	-	313.74
	F(%)		-	-	-	-	-	-	-	15.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	58.38	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.36	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.22	-	-	-	-	-	-	-	781.00
	SD		-	-	-	-	-	-	-	243.50
	F(%)		-	-	-	-	-	-	-	15.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.22	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.20	-	-	-	-	236.00	246.80	281.50	282.00
	SD		-	-	-	-	82.79	50.71	53.16	60.29
	F(%)		-	-	-	-	95.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	58.18	-	-	-	-	-	12.31	5.95	6.90
	SD		-	-	-	-	-	11.48	6.21	6.76

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	58.16	-	-	-	-	-	-	-	742.00
	SD		-	-	-	-	-	-	-	344.42
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.10	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.04	-	-	-	-	-	-	-	432.00
	SD		-	-	-	-	-	-	-	33.82
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.80	-	-	-	-	-	-	-	-
	SD		-	-	-	-	-	-	-	-
	F(%)		-	-	-	-	-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.78	-	-	-	-	-	-	-	644.00
	SD		-	-	-	-	-	-	-	110.07
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.68	-	-	-	-	-	259.62	199.70	211.00
	SD		-	-	-	-	-	112.31	96.10	105.53
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.54	-	-	-	-	4.95	3.95	4.65	4.40
	SD		-	-	-	-	2.94	3.63	2.99	2.82
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.42	-	-	-	-	392.00	139.38	76.55	63.85
	SD		-	-	-	-	0.00	116.14	68.72	46.48
	F(%)		-	-	-	-	5.00	65.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.42	-	-	-	5.20	14.20	19.70	26.60	27.25
	SD		-	-	-	2.64	4.52	5.35	7.72	7.86
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.36	-	-	5.35	26.50	64.55	88.80	114.45	122.85

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	SD		-	-	1.39	6.22	12.40	15.15	24.31	23.15
	F(%)		-	-	100.00	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.32	-	-	-	-	-	38.09	37.15	45.90
	SD		-	-	-	-	-	38.30	41.24	36.42
	F(%)		-	-	-	-	-	55.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.24	-	-	-	-	68.30	94.65	124.85	144.55
	SD		-	-	-	-	54.98	50.27	48.21	40.09
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.12	-	-	-	-	-	232.64	203.65	200.40
	SD		-	-	-	-	-	114.15	83.38	66.36
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.04	-	-	-	-	149.00	132.70	144.85	154.00
	SD		-	-	-	-	69.37	60.95	40.98	35.09
	F(%)		-	-	-	-	75.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.00	-	-	-	-	-	-	1074.00	471.33
	SD		-	-	-	-	-	-	0.00	350.48
	F(%)		-	-	-	-	-	-	5.00	15.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	56.88	-	-	-	-	-	-	-	613.00
	SD		-	-	-	-	-	-	-	218.00
	F(%)		-	-	-	-	-	-	-	10.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.64	-	-	-	-	-	121.56	107.40	96.25
	SD		-	-	-	-	-	64.78	60.13	53.62
	F(%)		-	-	-	-	-	80.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	56.64	-	-	3.00	4.75	13.30	19.75	28.40	27.00
	SD		-	-	0.00	3.36	5.09	5.41	7.21	5.37
	F(%)		-	-	5.00	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.62	-	-	-	-	-	-	-	347.90
	SD		-	-	-	-	-	-	-	177.39
	F(%)		-	-	-	-	-	-	-	50.00

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.58	-	-	-	-	-	121.80	101.85	91.70
	SD		-	-	-	-	-	54.80	51.03	38.90
	F(%)		-	-	-	-	-	75.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.58	-	-	-	-	-	375.42	274.30	285.05
	SD		-	-	-	-	-	180.22	139.56	166.82
	F(%)		-	-	-	-	-	60.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	56.52	-	-	-	-	-	773.90	799.10	883.90
	SD		-	-	-	-	-	190.19	183.03	107.67
	F(%)		-	-	-	-	-	50.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	56.52	-	-	-	-	-	220.64	265.45	269.70
	SD		-	-	-	-	-	77.46	122.68	120.29
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.44	-	-	-	-	-	59.60	41.90	45.10
	SD		-	-	-	-	-	41.59	23.26	33.87
	F(%)		-	-	-	-	-	50.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.30	-	-	-	-	-	87.93	73.40	69.35
	SD		-	-	-	-	-	45.55	48.89	51.35
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.28	-	-	-	-	-	-	-	395.33
	SD		-	-	-	-	-	-	-	141.40
	F(%)		-	-	-	-	-	-	-	15.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.16	-	-	-	-	-	-	-	410.22
	SD		-	-	-	-	-	-	-	198.93
	F(%)		-	-	-	-	-	-	-	45.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.12	-	-	-	-	-	26.73	22.55	24.05
	SD		-	-	-	-	-	20.78	13.89	12.25
	F(%)		-	-	-	-	-	75.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	55.92	-	-	-	-	-	-	-	536.50
	SD		-	-	-	-	-	-	-	297.79

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.84	-	-	-	-	2.85	3.55	5.05	4.65
	SD		-	-	-	-	2.17	2.65	2.85	2.90
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	55.78	-	-	-	-	-	134.56	131.15	121.70
	SD		-	-	-	-	-	77.69	70.14	61.83
	F(%)		-	-	-	-	-	90.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.46	-	-	-	-	-	98.21	66.95	69.30
	SD		-	-	-	-	-	76.81	45.75	43.84
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	55.46	-	-	-	20.00	4.95	6.75	9.95	8.25
	SD		-	-	-	0.00	4.20	4.23	5.35	3.74
	F(%)		-	-	-	5.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.44	-	-	5.75	26.95	69.95	90.25	107.55	110.50
	SD		-	-	1.61	4.96	9.81	12.09	15.73	16.83
	F(%)		-	-	100.00	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.02	-	-	-	-	-	23.47	29.15	25.75
	SD		-	-	-	-	-	18.22	20.60	19.19
	F(%)		-	-	-	-	-	85.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.92	-	-	-	-	67.50	75.00	81.35	78.85
	SD		-	-	-	-	35.98	36.74	31.24	24.03
	F(%)		-	-	-	-	90.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	54.86	-	-	-	2.90	7.30	9.50	13.65	11.70
	SD		-	-	-	2.53	7.34	8.51	10.73	10.34
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.78	-	-	-	-	79.85	86.15	95.95	95.15
	SD		-	-	-	-	45.45	37.52	35.87	33.83
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.74	-	-	-	-	-	39.50	30.60	28.70

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	SD		-	-	-	-	-	27.67	32.06	24.14
	F(%)		-	-	-	-	-	60.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.32	-	-	-	-	-	-	-	197.60
	SD		-	-	-	-	-	-	-	70.41
	F(%)		-	-	-	-	-	-	-	25.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	54.28	-	-	-	-	-	-	-	599.20
	SD		-	-	-	-	-	-	-	174.72
	F(%)		-	-	-	-	-	-	-	25.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	54.00	-	-	7.00	7.50	19.15	25.70	30.85	30.00
	SD		-	-	0.00	4.02	6.36	8.98	9.11	7.77
	F(%)		-	-	5.00	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	54.00	-	-	-	-	-	366.46	274.70	274.20
	SD		-	-	-	-	-	183.74	168.22	143.98
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.88	-	-	-	-	-	-	886.00	380.44
	SD		-	-	-	-	-	-	131.59	273.51
	F(%)		-	-	-	-	-	-	15.00	45.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.68	-	-	-	-	-	-	-	300.00
	SD		-	-	-	-	-	-	-	104.66
	F(%)		-	-	-	-	-	-	-	35.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.64	-	-	-	-	-	-	-	276.00
	SD		-	-	-	-	-	-	-	6.00
	F(%)		-	-	-	-	-	-	-	10.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi$	MP	53.54	-	-	-	-	-	666.71	641.75	644.00
	SD		-	-	-	-	-	174.04	194.50	188.20
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.50	-	-	-	388.00	163.45	166.60	170.00	177.60
	SD		-	-	-	0.00	108.54	62.02	45.77	43.01
	F(%)		-	-	-	5.00	100.00	100.00	100.00	100.00

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	53.50	-	-	-	-	-	814.25	851.65	911.65
	SD		-	-	-	-	-	124.11	148.93	123.59
	F(%)		-	-	-	-	-	80.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.44	-	-	-	25.20	61.70	74.20	88.90	89.05
	SD		-	-	-	9.70	12.28	16.56	20.84	20.59
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	53.44	-	-	-	-	7.70	9.15	9.55	8.85
	SD		-	-	-	-	6.62	8.08	8.67	7.34
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.38	-	-	-	-	-	75.08	52.50	52.10
	SD		-	-	-	-	-	51.06	34.70	21.79
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi_1 \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.32	-	-	-	-	44.65	43.05	49.60	43.35
	SD		-	-	-	-	24.65	23.31	17.42	15.04
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.24	-	-	-	80.10	168.30	196.25	224.50	228.15
	SD		-	-	-	31.21	37.69	35.06	27.65	22.43
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.22	-	-	-	-	-	99.69	95.05	105.55
	SD		-	-	-	-	-	59.47	59.01	50.92
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.22	-	-	-	-	-	84.06	72.15	76.40
	SD		-	-	-	-	-	56.39	34.76	37.34
	F(%)		-	-	-	-	-	80.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	53.20	-	-	-	-	273.00	49.31	29.95	26.95
	SD		-	-	-	-	0.00	45.70	25.06	22.22
	F(%)		-	-	-	-	5.00	65.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.16	-	-	20.15	94.45	203.65	257.30	296.60	294.95
	SD		-	-	3.29	17.64	23.88	20.32	28.83	25.47

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	F(%)		-	-	100.00	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.12	-	-	-	-	50.80	53.75	62.75	58.65
	SD		-	-	-	-	22.03	18.40	20.90	15.12
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.12	-	-	-	-	-	442.11	272.30	277.90
	SD		-	-	-	-	-	196.35	183.59	144.53
	F(%)		-	-	-	-	-	45.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	53.08	-	-	-	-	-	42.83	31.30	26.90
	SD		-	-	-	-	-	46.47	34.06	25.64
	F(%)		-	-	-	-	-	60.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.08	-	-	-	-	249.75	261.95	303.55	290.25
	SD		-	-	-	-	139.00	108.57	100.51	80.67
	F(%)		-	-	-	-	80.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	52.98	-	-	-	-	-	-	-	437.50
	SD		-	-	-	-	-	-	-	77.50
	F(%)		-	-	-	-	-	-	-	10.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi$	MP	52.82	-	-	-	176.50	382.30	473.85	517.00	532.25
	SD		-	-	-	53.35	75.15	72.13	70.83	85.55
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.76	-	-	-	-	-	224.50	130.85	128.90
	SD		-	-	-	-	-	95.69	66.26	52.31
	F(%)		-	-	-	-	-	50.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	52.70	-	-	-	-	-	812.46	853.05	867.95
	SD		-	-	-	-	-	182.67	160.90	171.41
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.68	-	-	-	-	291.12	303.65	321.45	316.50
	SD		-	-	-	-	139.06	143.65	96.82	91.14
	F(%)		-	-	-	-	80.00	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	52.66	-	-	-	-	-	-	-	736.00

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	SD		-	-	-	-	-	-	-	115.00
	F(%)		-	-	-	-	-	-	-	10.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	52.66	-	-	1.00	5.95	15.35	21.95	30.40	30.05
	SD		-	-	0.00	1.99	5.71	7.85	11.44	12.13
	F(%)		-	-	100.00	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.58	-	-	-	-	14.00	13.15	16.30	14.95
	SD		-	-	-	-	9.37	6.81	7.30	7.10
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.50	-	-	-	-	-	-	-	645.75
	SD		-	-	-	-	-	-	-	239.52
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.50	-	-	-	-	-	-	-	835.33
	SD		-	-	-	-	-	-	-	248.20
	F(%)		-	-	-	-	-	-	-	15.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	52.42	-	-	-	-	-	303.36	286.00	295.05
	SD		-	-	-	-	-	191.38	194.30	145.66
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.32	-	-	-	-	-	294.62	245.70	234.25
	SD		-	-	-	-	-	115.33	103.45	67.87
	F(%)		-	-	-	-	-	80.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.30	-	-	-	-	-	-	-	751.83
	SD		-	-	-	-	-	-	-	275.14
	F(%)		-	-	-	-	-	-	-	30.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi$	MP	52.26	-	-	-	-	-	581.25	567.55	528.45
	SD		-	-	-	-	-	199.05	220.81	212.27
	F(%)		-	-	-	-	-	80.00	100.00	100.00
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.08	-	-	-	-	-	34.67	34.45	36.20
	SD		-	-	-	-	-	28.17	22.51	24.68
	F(%)		-	-	-	-	-	90.00	100.00	100.00

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.08	-	-	-	-	-	447.29	347.30	313.85
	SD		-	-	-	-	-	198.06	146.76	120.29
	F(%)		-	-	-	-	-	70.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.00	-	-	-	-	-	302.90	232.85	253.30
	SD		-	-	-	-	-	185.76	169.79	184.73
	F(%)		-	-	-	-	-	50.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	51.88	-	-	17.00	23.25	53.20	65.70	76.90	75.70
	SD		-	-	0.00	11.69	16.82	14.59	16.71	13.89
	F(%)		-	-	5.00	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.84	-	-	-	-	11.50	12.25	16.40	15.55
	SD		-	-	-	-	7.47	5.02	4.83	4.91
	F(%)		-	-	-	-	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi$	MP	51.82	-	-	-	73.10	160.75	196.10	219.05	214.65
	SD		-	-	-	31.43	43.83	39.15	33.52	34.95
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.82	-	-	-	-	-	431.73	287.75	283.85
	SD		-	-	-	-	-	198.96	119.58	106.68
	F(%)		-	-	-	-	-	55.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.80	-	-	-	-	-	-	-	803.00
	SD		-	-	-	-	-	-	-	229.84
	F(%)		-	-	-	-	-	-	-	20.00
$\varphi_1 \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.78	-	-	-	23.45	53.65	65.40	84.95	86.30
	SD		-	-	-	10.30	16.60	14.44	18.78	18.11
	F(%)		-	-	-	100.00	100.00	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	51.78	-	-	-	-	-	326.85	305.70	296.15
	SD		-	-	-	-	-	127.65	128.39	101.50
	F(%)		-	-	-	-	-	65.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.72	-	-	-	-	-	-	807.00	655.50
	SD		-	-	-	-	-	-	0.00	349.01

Continued on next page

Chain	Type	Perf	PTS	M-HH(5)	M-HH(10)	M-HH(15)	M-HH(20)	M-HH(25)	M-HH(30)	M-HH(35)
	F(%)		-	-	-	-	-	-	5.00	50.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	51.72	-	-	-	-	-	565.54	474.45	463.90
	SD		-	-	-	-	-	234.16	213.17	215.20
	F(%)		-	-	-	-	-	65.00	100.00	100.00

Table 8: Comparison position good chains for M-HH with different values of len_sub

D Comparison HHs Chain Positions

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	1527.00	-	554.83	383.79
	SD		-	-	287.09	220.95
	F (%)		-	-	30.00	95.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	1569.00	-	88.25	433.95
	SD		-	-	56.61	228.30
	F (%)		-	-	20.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	855.00	-	209.50	339.20
	SD		-	-	67.81	157.66
	F (%)		-	-	30.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	777.00	-	42.10	366.05
	SD		-	-	15.68	115.41
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	813.00	-	149.00	904.00
	SD		-	-	96.42	498.90
	F (%)		-	-	30.00	35.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	297.00	-	285.50	375.85
	SD		-	-	69.63	104.89
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	435.00	-	600.40	1314.57
	SD		-	-	97.40	86.58
	F (%)		-	-	25.00	35.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	60.00	1359.00	-	68.71	866.50
	SD		-	-	39.30	510.07
	F (%)		-	-	35.00	50.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	1107.00	-	4.55	355.80
	SD		-	-	4.47	209.49
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	327.00	-	549.33	474.45
	SD		-	-	175.06	314.77
	F (%)		-	-	15.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	1335.00	-	618.25	531.00
	SD		-	-	198.68	242.41
	F (%)		-	-	20.00	30.00
$\varphi_1 \xrightarrow{BT} \varphi$	MP	59.92	9.00	110.70	390.55	490.80
	SD		-	12.95	49.43	85.33
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.52	969.00	-	7.20	310.50
	SD		-	-	5.97	123.12
	F (%)		-	-	100.00	100.00

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.52	1311.00	117.15	422.85	876.15
	SD		-	50.35	70.59	253.94
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	59.32	63.00	-	973.70	471.95
	SD		-	-	49.74	125.59
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	59.28	1503.00	128.47	348.25	483.30
	SD		-	34.94	40.53	102.31
	F (%)		-	95.00	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.90	1245.00	-	447.53	274.85
	SD		-	-	193.71	137.65
	F (%)		-	-	85.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.86	1893.00	-	217.65	254.50
	SD		-	-	61.30	93.88
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.80	1671.00	-	399.28	557.80
	SD		-	-	189.47	268.18
	F (%)		-	-	90.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.78	1665.00	-	-	1355.00
	SD		-	-	-	87.00
	F (%)		-	-	-	10.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	58.68	1779.00	-	559.47	430.35
	SD		-	-	255.89	174.12
	F (%)		-	-	85.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	58.62	21.00	-	528.90	403.15
	SD		-	-	126.12	139.71
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	58.62	1515.00	-	751.40	488.40
	SD		-	-	237.40	181.13
	F (%)		-	-	25.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	58.60	2475.00	-	-	971.67
	SD		-	-	-	430.11
	F (%)		-	-	-	15.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.58	1719.00	-	-	-
	SD		-	-	-	-
	F (%)		-	-	-	-
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.54	1581.00	-	421.50	837.55
	SD		-	-	162.29	384.54
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.52	1803.00	-	-	739.93
	SD		-	-	-	412.16

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
	F (%)		-	-	-	75.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.50	2457.00	-	-	382.95
	SD		-	-	-	326.30
	F (%)		-	-	-	95.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	58.50	1521.00	-	1040.33	413.45
	SD		-	-	75.83	200.12
	F (%)		-	-	15.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	58.46	1329.00	-	1104.17	398.65
	SD		-	-	75.62	260.54
	F (%)		-	-	30.00	85.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.46	1863.00	-	578.05	-
	SD		-	-	224.08	-
	F (%)		-	-	100.00	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.42	1737.00	-	566.00	474.85
	SD		-	-	313.74	211.78
	F (%)		-	-	15.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	58.38	1773.00	-	-	509.72
	SD		-	-	-	332.74
	F (%)		-	-	-	90.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.36	2013.00	-	-	-
	SD		-	-	-	-
	F (%)		-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.22	2025.00	-	781.00	362.30
	SD		-	-	243.50	208.30
	F (%)		-	-	15.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.22	1695.00	-	-	-
	SD		-	-	-	-
	F (%)		-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.20	1815.00	-	282.00	340.00
	SD		-	-	60.29	57.32
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	58.18	781.00	-	6.90	505.30
	SD		-	-	6.76	161.22
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	58.16	1323.00	-	742.00	480.89
	SD		-	-	344.42	353.72
	F (%)		-	-	20.00	95.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.10	1713.00	-	-	821.88
	SD		-	-	-	379.58
	F (%)		-	-	-	40.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	58.04	1923.00	-	432.00	870.84

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
	SD		-	-	33.82	408.15
	F (%)		-	-	20.00	95.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.80	2019.00	-	-	-
	SD		-	-	-	-
	F (%)		-	-	-	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.78	1917.00	-	644.00	678.62
	SD		-	-	110.07	436.69
	F (%)		-	-	20.00	80.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.68	1065.00	-	211.00	329.80
	SD		-	-	105.53	113.04
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.54	993.00	-	4.40	438.15
	SD		-	-	2.82	168.46
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.42	1419.00	-	63.85	502.45
	SD		-	-	46.48	181.75
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.42	999.00	5.20	27.25	431.45
	SD		-	2.64	7.86	133.26
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.36	1041.00	26.50	122.85	438.90
	SD		-	6.22	23.15	102.55
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.32	1431.00	-	45.90	678.25
	SD		-	-	36.42	413.41
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.24	1215.00	-	144.55	737.50
	SD		-	-	40.09	332.99
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.12	225.00	-	200.40	346.80
	SD		-	-	66.36	92.45
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.04	1191.00	-	154.00	489.85
	SD		-	-	35.09	132.34
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	57.00	1155.00	-	471.33	522.84
	SD		-	-	350.48	347.52
	F (%)		-	-	15.00	95.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	56.88	1731.00	-	613.00	547.65
	SD		-	-	218.00	274.68
	F (%)		-	-	10.00	100.00

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.64	1401.00	-	96.25	346.75
	SD		-	-	53.62	140.87
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	56.64	1045.00	4.75	27.00	608.05
	SD		-	3.36	5.37	154.83
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.62	1449.00	-	347.90	-
	SD		-	-	177.39	-
	F (%)		-	-	50.00	-
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.58	453.00	-	91.70	408.65
	SD		-	-	38.90	243.13
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.58	513.00	-	285.05	490.75
	SD		-	-	166.82	200.96
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	56.52	1059.00	-	883.90	516.70
	SD		-	-	107.67	177.98
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	56.52	1053.00	-	269.70	427.65
	SD		-	-	120.29	144.34
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.44	1413.00	-	45.10	387.85
	SD		-	-	33.87	162.32
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.30	1035.00	-	69.35	308.45
	SD		-	-	51.35	115.54
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.28	1023.00	-	395.33	276.75
	SD		-	-	141.40	165.69
	F (%)		-	-	15.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.16	1443.00	-	410.22	975.53
	SD		-	-	198.93	274.56
	F (%)		-	-	45.00	75.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	56.12	519.00	-	24.05	442.60
	SD		-	-	12.25	201.03
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	55.92	1425.00	-	536.50	383.10
	SD		-	-	297.79	229.13
	F (%)		-	-	20.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.84	1003.00	-	4.65	602.80
	SD		-	-	2.90	177.48

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	55.78	2223.00	-	121.70	839.21
	SD		-	-	61.83	399.37
	F (%)		-	-	100.00	95.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.46	301.00	-	69.30	547.25
	SD		-	-	43.84	159.25
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	55.46	1051.00	20.00	8.25	721.05
	SD		-	0.00	3.74	157.48
	F (%)		-	5.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.44	13.00	26.95	110.50	659.30
	SD		-	4.96	16.83	146.20
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	55.02	1219.00	-	25.75	775.95
	SD		-	-	19.19	397.16
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.92	1507.00	-	78.85	768.15
	SD		-	-	24.03	157.11
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	54.86	55.00	2.90	11.70	986.80
	SD		-	2.53	10.34	216.53
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.78	1315.00	-	95.15	882.35
	SD		-	-	33.83	239.52
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.74	1195.00	-	28.70	696.60
	SD		-	-	24.14	221.14
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	54.32	1585.00	-	197.60	895.94
	SD		-	-	70.41	417.35
	F (%)		-	-	25.00	90.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	54.28	27.00	-	599.20	592.95
	SD		-	-	174.72	199.63
	F (%)		-	-	25.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	54.00	19.00	7.50	30.00	747.10
	SD		-	4.02	7.77	159.65
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	54.00	339.00	-	274.20	416.70
	SD		-	-	143.98	198.44
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.88	1867.00	-	380.44	-

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
	SD		-	-	273.51	-
	F (%)		-	-	45.00	-
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.68	1249.00	-	300.00	501.75
	SD		-	-	104.66	235.29
	F (%)		-	-	35.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.64	1783.00	-	276.00	634.20
	SD		-	-	6.00	230.35
	F (%)		-	-	10.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi$	MP	53.54	782.00	-	644.00	1133.30
	SD		-	-	188.20	237.33
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.50	231.00	388.00	177.60	488.10
	SD		-	0.00	43.01	106.67
	F (%)		-	5.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	53.50	39.00	-	911.65	510.25
	SD		-	-	123.59	115.26
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.44	825.00	25.20	89.05	373.35
	SD		-	9.70	20.59	109.51
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	53.44	31.00	-	8.85	827.15
	SD		-	-	7.34	194.88
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.38	1897.00	-	52.10	483.00
	SD		-	-	21.79	155.89
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.32	1491.00	-	43.35	474.00
	SD		-	-	15.04	154.27
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.24	399.00	80.10	228.15	493.20
	SD		-	31.21	22.43	91.72
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.22	25.00	-	105.55	683.45
	SD		-	-	50.92	187.21
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	53.22	1819.00	-	76.40	545.45
	SD		-	-	37.34	145.07
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	53.20	1513.00	-	26.95	841.35
	SD		-	-	22.22	184.63
	F (%)		-	-	100.00	100.00

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.16	489.00	94.45	294.95	509.65
	SD		-	17.64	25.47	88.80
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.12	483.00	-	58.65	399.30
	SD		-	-	15.12	111.13
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.12	561.00	-	277.90	454.25
	SD		-	-	144.53	128.82
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	53.08	1321.00	-	26.90	801.45
	SD		-	-	25.64	311.32
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	53.08	693.00	-	290.25	865.40
	SD		-	-	80.67	309.62
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	52.98	1675.00	-	437.50	629.85
	SD		-	-	77.50	379.42
	F (%)		-	-	10.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi$	MP	52.82	20.00	176.50	532.25	844.45
	SD		-	53.35	85.55	135.42
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.76	801.00	-	128.90	480.15
	SD		-	-	52.31	159.39
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi$	MP	52.70	507.00	-	867.95	529.05
	SD		-	-	171.41	139.72
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.68	663.00	-	316.50	431.70
	SD		-	-	91.14	114.42
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi$	MP	52.66	771.00	-	736.00	324.80
	SD		-	-	115.00	170.71
	F (%)		-	-	10.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi$	MP	52.66	49.00	5.95	30.05	981.85
	SD		-	1.99	12.13	187.24
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.58	819.00	-	14.95	431.70
	SD		-	-	7.10	146.80
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.50	885.00	-	645.75	371.20
	SD		-	-	239.52	199.48

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
	F (%)		-	-	20.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.50	741.00	-	835.33	280.00
	SD		-	-	248.20	228.42
	F (%)		-	-	15.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	52.42	33.00	-	295.05	716.05
	SD		-	-	145.66	155.13
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.32	729.00	-	234.25	426.10
	SD		-	-	67.87	110.02
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.30	687.00	-	751.83	1214.38
	SD		-	-	275.14	109.63
	F (%)		-	-	30.00	40.00
$\varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi$	MP	52.26	1322.00	-	528.45	726.55
	SD		-	-	212.27	284.47
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.08	2709.00	-	36.20	236.00
	SD		-	-	24.68	186.15
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.08	2679.00	-	313.85	211.60
	SD		-	-	120.29	249.55
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{RM} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	52.00	429.00	-	253.30	827.15
	SD		-	-	184.73	346.02
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	51.88	493.00	23.25	75.70	709.15
	SD		-	11.69	13.89	147.62
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.84	2469.00	-	15.55	256.50
	SD		-	-	4.91	86.51
	F (%)		-	-	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi$	MP	51.82	61.00	73.10	214.65	982.45
	SD		-	31.43	34.95	110.16
	F (%)		-	100.00	100.00	100.00
$\varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.82	765.00	-	283.85	358.45
	SD		-	-	106.68	104.31
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.80	615.00	-	803.00	633.75
	SD		-	-	229.84	404.55
	F (%)		-	-	20.00	100.00
$\varphi_1 \xrightarrow{CM} \varphi \xrightarrow{RM} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.78	1839.00	23.45	86.30	205.85

Continued on next page

Chain	Type	Perf	PTS	M-HH (15)	M-HH (35)	PM-HH
	SD		-	10.30	18.11	73.21
	F (%)		-	100.00	100.00	100.00
$\varphi_1 \xrightarrow{BT} \varphi \xrightarrow{CM} \varphi \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi$	MP	51.78	67.00	-	296.15	835.65
	SD		-	-	101.50	163.81
	F (%)		-	-	100.00	100.00
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi_1 \xrightarrow{BT} \varphi$	MP	51.72	747.00	-	655.50	-
	SD		-	-	349.01	-
	F (%)		-	-	50.00	-
$\varphi \xrightarrow{CM} \varphi_1 \xrightarrow{BT} \varphi \xrightarrow{RM} \varphi \xrightarrow{BT} \varphi$	MP	51.72	501.00	-	463.90	364.90
	SD		-	-	215.20	149.33
	F (%)		-	-	100.00	100.00

Table 9: Comparison of position chains for different HHs