CHARACTERISING FITNESS LANDSCAPES WITH FITNESS-PROBABILITY CLOUD AND ITS APPLICATIONS TO ALGORITHM CONFIGURATION

by
GUANZHOU LU

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Science
University of Birmingham
August 2013

Abstract

Metaheuristics are approximation optimisation techniques widely applied to solve complex optimisation problems. Since metaheuristics are general algorithmic frameworks, a metaheuristic algorithm can have many variants if different configurations (e.g., choice of search operator, numerical parameters, etc.) are applied. In particular, it is widely acknowledged that finding good algorithm configurations is essential to obtain robust and high algorithm performance. Despite a large number of developed metaheuristic algorithms, a limited amount of work has been done to understand on which kinds of problems the proposed algorithm will perform well or poorly and why. Given this lack of understanding of the problem difficulty in relation to algorithms, fundamental questions like how to select and configure the best suited algorithm for solving a particular problem instance remain unanswered. A useful solution to this dilemma is to use fitness landscape analysis to gain an in-depth understanding of which algorithms, or algorithm variants are best suited for solving which kinds of problem instances, even to dynamically determine the best algorithm configuration during different stages of a search algorithm.

This thesis for the first time bridges the gap between fitness landscape analysis and algorithm configuration, i.e., finding the best suited configuration of a given algorithm for solving a particular problem instance. In particular, this thesis addresses three prominent issues in applying fitness landscape analysis to build enhanced techniques for algorithm configuration. First, there is a lack of an effective approach for fitness landscape analysis that can be computed efficiently and used to guide search. Second, in the static algorithm configuration, there is a lack of a generic approach to automatically determine a priori the best suited configuration of a given algorithm on a per-instance base. Third, in the dynamic algorithm configuration, there is a lack of predictive methods to determine the optimal configuration during the search process based on the expected performance instead of past performance only. Studies in this thesis contribute to the following:

a. Developing a novel and effective approach to characterise fitness landscapes and measure problem difficulty with respect to algorithms.

b. Incorporating fitness landscape analysis in building a generic (problem-independent) approach, which can perform automatic algorithm configuration on a per-instance base, and in designing novel and effective algorithm configurations. Results from the case studies show that the

developed automatic algorithm configuration method significantly outperforms the state-of-the-art in automatically configuring the $(\mu + \lambda)$ EAs for solving the unique input output sequence problem. Results also show that the local search heuristic developed using results from fitness landscape analysis significantly outperforms a stochastic hill climber and the state-of-the-art in solving the next release problem in software engineering.

c. Incorporating fitness landscape analysis in establishing a generic framework for designing adaptive heuristic algorithms. Results from the case study, where the framework is applied to develop an adaptive heuristic algorithm for the minimum vertex cover problem (MVC), show that the proposed algorithm either outperforms or is comparable to the state-of-the-art algorithms for MVC on both well-studied benchmarks and real-world instances. New lower bounds have been found by the proposed algorithm on several hard problem instances.

Acknowledgements

First of all, my sincere thanks to Professor Xin Yao for being an excellent guide throughout my PhD studies, continually providing constructive advice and always encouraging me to explore novel routes. His advice has helped my research achieve the balance between exploration and exploitation.

I am also very grateful to the many others who have helped me along the way. Many thanks go to Dr John Bullinaria and Dr Rami Bahsoon for being members of my thesis group and their regular insight and perspective on my research, and to Dr. Jinlong Li for the collaborative work and help over the last few years. Thanks also to those who have been involved in discussions that have helped shape this thesis: Peter Lewis, Leandro Minku, Huanhuan Chen, Haobo Fu, Shuo Wang, Khulood Alyahya, Ayush Joshi, Christine Zarges, and many other former and current members of CERCIA. Additionally, thanks to those involved in the EPSRC Software Engineering By Automated SEarch project for sharing their thoughts and expertise in different domains.

Finally, my special thanks to my parents, my fiancee Lan Xu, and all my families. Indeed more thanks than I could express for their continues love and support.

Publications arising from this Thesis

- [1] G. Lu, R. Bahsoon, and X. Yao, "Applying elementary landscape analysis to search-based software engineering," In *Proceedings of the 2nd International Symposium on Search Based Software Engineering, SSBSE'10*, pages 3-8, Washington, DC, USA, 2010. IEEE Computer Society
- [2] G. Lu, J. Li, and X. Yao. "Fitness-probability cloud and a measure of problem hardness for evolutionary algorithms," In *Peter Merz and Jin-Kao Hao, editors, Evolutionary Computation in Combinatorial Optimization, volume 6622 of Lecture Notes in Computer Science*, pages 108-117. Springer Berlin Heidelberg, 2011.
- [3] J. Li, G. Lu, and X. Yao. "Fitness Landscape-based Parameter Tuning Method for Evolutionary Algorithms for Computing Unique Input Output Sequences," In *Proceedings of the 18th international conference on Neural Information Processing - Volume Part II, ICONIP'11*, pages 453-460, Berlin, Heidelberg, 2011. Springer-Verlag.
- [4] G. Lu, J. Li, and X. Yao. "Fitness Landscape and Problem Difficulty in Evolutionary Algorithms: From Theory to Applications," In *Hendrik Richter, Andries Engelbrecht, editors: Recent advances in the theory and application of fitness landscapes*, Springer-Verlag, 2013.
- [5] G. Lu, J. Li, and X. Yao. "Adaptive Heuristic Algorithms by Predicting Search Difficulties," *submitted to Artificial Intelligence*, 2013.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Fitness Landscape Analysis in Combinatorial Optimisation

In recent years, metaheuristics such as Evolutionary Algorithm (EA), Stochastic Local Search (SLS), Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO) and others are emerging as successful approaches for solving complex optimisation problems where classical exact methods are either infeasible or perform poorly. This is due to the fact that many optimisation problems are NP-hard [43], which can require exponential time for exact methods to solve. Nevertheless, metaheuristics are to find good solutions within reasonable time without the guarantee of finding the optimal solutions. Moreover, metaheuristics often require little knowledge about the optimisation problems being solved such as the gradient information, and therefore can be applied to solve a wide spectrum of optimisation problems where the objective function is not differentiable or can only be computed by simulation.

Metaheuristics are a general class of optimisation techniques which include many different approaches, e.g., local search algorithms such as Tabu Search (TS), Stochastic Local Search (SLS) and Simulated Annealing (SA); population-based algorithms such as Evolutionary Algorithm (EA), Ant Colony Optimization (ACO). In addition, metaheuristics are general algorithmic frameworks which potentially have many different variants under different configurations (e.g., choice of search operator/heuristic, parameter settings, etc.). For solving an optimisation problem with metaheuristics, a problem naturally arises of how to select and configure an appropriate algorithm.

Over the years the research focus in metaheuristics has largely been on the algorithmic side. Many different metaheuristics have been developed which empirically demonstrated the effectiveness on one or more optimisation problems, or failed on some other problems. In contrast, relatively little attention has been paid to study the implications behind the empirical results, i.e., on understanding which kinds of problems the proposed algorithm will perform well or poorly and why. Given this lack of understanding of problem difficulty in relation to algorithms, despite many successes of metaheuristics in solving complex optimisation problems, fundamental questions like how to select and configure the most effective algorithm for solving a given problem remain unanswered. Recent theoretical investigations such as runtime analysis [68] have made some progress in addressing this problem, where on particular problem instances the EA-hardness [58], when a large population is useful [56] and the interactions between mutation and selection [77] are investigated. However, in practice there is in fact very limited understanding of which algorithms, or algorithm variants are best suited for solving which kinds of problems. The most common technique for selecting and configuring an appropriate algorithm for solving a given problem is by trial and error, as expressed by Culberson [26]: "The researcher trying to solve a problem is then placed in the unfortunate position of having to find a representation, operators and parameter settings to make a poorly understood system solve a poorly understood problem. In many cases he might be better served concentrating on the problem itself".

To overcome this long standing challenge, the ideal would be to have a single best algorithm for every problem. However, this is infeasible as the well-known no free lunch theorem [133] for optimisation has concluded that no one optimisation algorithm is superior to the other on all problems. Furthermore, it is widely acknowledged that finding good algorithm configurations are essential to obtain robust and high algorithm performance. Therefore, instead of finding the best algorithm in general, the problem becomes that of finding the best suited configuration of a given algorithm for solving a particular problem. This problem is referred to as the algorithm configuration problem [66] throughout this thesis. It is worth noting that while some work refers to setting numerical parameters of metaheuristics as parameter tuning, algorithm configuration concerns not only the problem of tuning numerical parameters, but selecting and combining discrete building blocks (e.g., categorical parameters such as choice of search operator/heuristic) to build an effective algorithm.

The algorithm configuration problem has been studied extensively, particularly in selecting appropriate configurations automatically. This idea was introduced in [138] where two mutation operators were used in the Improved Fast Evolutionary Programming. After this seminal work, the studies of automatic algorithm configuration proceed in two routes. Static algorithm configuration attempts to determine a priori the optimal configuration of a given algorithm for solving a particular problem, before applying the algorithm [1, 12, 66, 89, 137]. In contrast, dynamic algorithm configuration tends to determine an optimal time-varying schedule for applying different algorithm configurations during the search process [24, 29, 45, 59, 71, 116, 128]. Despite a large number of proposed techniques for tackling both the static and dynamic algorithm configuration problems, as of yet no satisfactory technique has been developed .

A useful solution to this dilemma is to use fitness landscape analysis to gain an in-depth understanding of which algorithms, or algorithm variants are best suited for solving which kinds of problems. The notion of fitness landscapes, originally proposed in [134], underlies a large body of work in problem difficulty studies with reference to metaheuristics. The fitness landscape analysis can be regarded as a powerful analytical tool, particularly in understanding characteristics of optimisation problems and the associated behaviours of metaheuristics in optimising them. Typically a fitness landscape is defined under the notion of neighbourhood/distance, therefore the same fitness function can have many different fitness landscapes defined under different neighbourhood operators. Formally, the fitness landscape is defined as a triple (X, N, f), where X is a set of candidate solutions, the objective function $f : X \longmapsto \Re$ assigns a real-valued fitness to each solution in X and the neighbourhood operator $N : x \longmapsto N(x)$ imposes a neighbourhood structure among X. Given a candidate solution $x \in X, N(x)$ is the neighbourhood set that are obtained by applying one step of the neighbourhood operator.

Many studies on fitness landscape analysis for in-depth understanding of problems in relation to algorithms exist in the literature. The existing approaches for fitness landscape analysis proceed along two main routes. On one hand, the qualitative approaches [14, 15, 30, 38, 44, 63, 120] focussed on describing which characteristics make the fitness landscapes hard to optimise and which do not. On the other hand, the quantitative approaches [28, 70, 84, 101, 119] defined an algebraic measure to explicitly quantify the difficulty of fitness landscapes.

This thesis identifies the fundamental link between fitness landscape analysis and algorithm configuration, in a sense that the in-depth understanding of problem difficulty in relation to algorithms gained through fitness landscape analysis should naturally be of help in determining the optimal configuration of a given algorithm for solving a particular problem.

Although fitness landscape analysis shows great promise to address the algorithm configuration problem, there are three important research issues to be fully addressed:

- First of all, despite a large number of techniques developed for fitness landscape analysis, very few techniques are used in practice. This is due to the fact that the developed techniques are either proved to be unreliable or unable to be used in practice due to their own limitations (e.g. require the global optima to be known). There is a lack of a reliable and effective approach for fitness landscape analysis, which can potentially be incorporated to directly address the algorithm configuration problem.
- In static algorithm configuration, most automatic algorithm configuration methods either produce a one-size-fits-all configuration of an algorithm for an entire set of instances [66] or perform per-instance configuration for a particular problem by making use of problem-specific features. There is a lack of a generic (problem-independent) approach to automatically find the optimal configuration for a given algorithm on a per-instance base.
- In dynamic algorithm configuration, the optimal configuration of a heuristic algorithm is determined based on utilisation of historical information only under the assumption that a configuration that performed well in the past is bound to perform well in the future. There is a lack of a predictive method which can determine the optimal configuration dynamically based on the expected performance of candidate configurations instead of past performance only.

This thesis aims to address these three important issues, particularly in developing a novel and effective approach for fitness landscape analysis. In particular, this thesis proposes for the first time to bridge the gap between fitness landscape analysis and algorithm configuration through explicitly applying results from fitness landscape analysis to address two prominent issues in both static and dynamic algorithm configuration, and design of novel and effective heuristics. The main contributions and the organisation of this thesis are presented in the following sections.

1.2 Major Contributions

1.2.1 A Novel Approach for Characterising Fitness Landscapes and Measuring Problem Difficulty

The first and foremost contribution of this thesis relates to the development of a novel, effective approach arising from theoretical investigations for characterising fitness landscapes and measuring problem difficulty. This contribution also establishes the foundations of other contributions in this thesis. Metaheuristics have been widely applied to solve hard optimisation problems, but the difficulty of selecting and configuring the best algorithm remains a huge challenge for practitioners. Fitness landscape analysis is to overcome this challenge by means of better understanding which algorithms, or algorithm variants are best suited for solving which kinds of problems. Although a large number of studies have been conducted to characterise fitness landscapes and measure problem difficulty with respect to metaheuristics, most developed techniques are either proved to be unreliable or unable to be used in practice due to their own limitations (e.g. require the global optima to be known), and there is a lack of a reliable and effective approach for characterising fitness landscapes and measuring problem difficulty. This thesis addresses this need through proposing a novel approach arising from theoretical investigations related to time complexity studies of metaheuristics for characterising fitness landscapes, where a predictive measure can be derived to explicitly quantify the problem difficulty with respect to algorithms.

The notion of escape probability is formally defined and a theoretical analysis is performed to investigate the relationship between the escape probability and the expected runtime which is usually taken as a difficulty measure in time complexity studies of metaheuristics. The results suggest that the escape probability is a critical factor in determining the expected runtime. Two further developments are obtained based on the escape probability. First, the fitness-probability cloud [82] is defined to obtain an overall characterisation of fitness landscapes. Second, a predictive problem difficulty measure, the accumulated escape probability, is derived from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms [82]. This generic measure can be used irrespective of the choice of algorithm or the problem of interest.

1.2.2 Incorporating Fitness Landscape Analysis for Static Algorithm Configuration

The second contribution of this thesis relates to incorporating the insight obtained by fitness landscape analysis to build a generic (problem-independent) approach to automatically find the optimal configuration for a given algorithm on a per-instance basis. Since metaheuristics are general algorithmic frameworks which potentially have many different variants under different configurations (e.g., choice of search operator/heuristic, parameter settings, etc.), it is widely acknowledged that finding good configurations is essential to obtain robust and high performance of metaheuristics. Furthermore, it is observed that a metaheuristic algorithm requires different configurations in order to find good solutions for different problem instances [41, 94, 137]. The problem of finding good configurations, also referred to as the static algorithm configuration problem, concerns determining a priori the most effective configuration of an algorithm for solving a particular problem instance. This problem is traditionally formulated as an optimisation problem, but unlike standard optimisation problems, the static algorithm configuration problem cannot be optimised directly due to the cost function to evaluate an algorithm configuration cannot be written analytically and is typically highly non-linear and very expensive to compute. Despite a number of approaches developed to tackle the static algorithm configuration problem, most previous approaches either only produce a one-size-fits-all configuration of the target algorithm for an entire set of problem instances [66] or perform per-instance configuration only for a particular problem by making use of problem-specific features [137]. There is a lack of a generic (problem-independent) approach to automatically find the optimal configuration for a given algorithm on a per-instance basis.

This thesis addresses this need in two steps. First, the static algorithm configuration problem is reformulated as a decision problem, which is to determine whether a configuration of the target algorithm is suitable for solving a particular problem instance. This reformulation is in the interest of practice since it is usually infeasible to explore the entire configuration space for finding the optimal configuration, it is somehow sufficient if an approximation solution can be efficiently identified within a finite set of candidate configurations. Second, the problem difficulty measure, the accumulated escape probability, is incorporated to build a generic approach which performs automatic algorithm configuration on a per-instance base. The proposed approach is based on learning the pattern which governs the relationship between the algorithm configurations and the characteristics of problem instances, where the characteristics of problem instances are extracted using the accumulated escape probability [78, 81].

Results from the case study on the unique input output sequence problem (UIO) show that the proposed approach can reliably determine whether a candidate configuration of the target algorithm, $(\mu+\lambda)$ EA, is suitable for solving a UIO instance, in terms of the pre-defined performance metric [78, 81]. More importantly, for a finite set of candidate configurations of the target algorithm, the proposed approach significantly outperforms a state-of-the-art automatic algorithm configuration method, ParamILS [66], in finding effective configurations of the target algorithm for solving the UIO problem.

Further to determining the best suited configuration within a finite set of candidate configurations, it is always useful to produce a novel, problem-specific configuration such as a new search operator or heuristic which potentially outperforms existing configurations for a particular class of problem instances. The design of such an effective configuration must account for the characteristics of problem instances. This thesis establishes a link between fitness landscape analysis and design of novel algorithm configuration, through proposing an approach to explicitly apply the theoretical results on elementary landscape to design novel and effective local search heuristics [80]. Results from the case study on the next release problem (NRP) in software engineering confirms the utility of the proposed approach, where the proposed elementary hill climbing algorithm developed using results from elementary landscape analysis significantly outperforms both the standard stochastic hill climber and the state-of-the-art in solving the NRP [80].

1.2.3 Incorporating Fitness Landscape Analysis for Dynamic Algorithm Configuration

The third contribution of this thesis relates to incorporating the insight gained through fitness landscape analysis to build enhanced techniques to address the dynamic algorithm configuration problem. Traditionally algorithm configuration methods determine a priori the most appropriate configuration of the target algorithm for solving a particular problem instance. However, there are both empirical and theoretical evidence showing that the most effective configuration of a given algorithm for solving a particular problem instance can vary during the search process

[116]. This motivates the development of heuristic algorithms that dynamically adapt their configurations (search operators, numerical parameters, etc.) during the search process. In particular, the behaviours of such heuristic algorithms are adapted to specific characteristics of problem instances, the generality of such algorithms would be improved to be able to solve a broad class of problems with diverse characteristics.

The design of adaptive heuristic algorithms concerns two main issues, the credit assignment mechanism and the selection mechanism. The former assigns a reward to a configuration based on evaluating the contribution of the configuration to the overall performance and the later serves as a selection rule in charge of selecting the configuration to use. It is noted that most existing adaptive heuristic algorithms perform adaptations based on the utilisation of historical information (past behaviours) only under the assumption that a configuration performed well in the past is bound to perform well in the future. However, considering only the past performance can be misleading, as the optimal fitness assigned to a configuration is a dynamic random variable and the underlying distribution of this random variable changes as the search proceeds.

This thesis addresses this important issue through incorporating predictive information provided by the predictive problem difficulty measure to design a systematic mechanism for adaptive heuristic search. The predictive measure originates from the fitness landscape analysis and explicitly quantifies problem difficulty with respect to algorithms. Equivalently the measure can be used to quantify the predicted performance of a specific algorithm for solving a given problem instance. A generic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure is proposed. To demonstrate the effectiveness of the proposed approach, the proposed framework is applied to design the fitness landscape based adaptive search algorithm (FAS) for tackling the minimum vertex cover problem

(MVC). FAS either outperforms or is comparable to the state-of-the-art algorithms for MVC on both well-studied benchmarks and real-world instances. New lower bounds have been found by FAS on several hard problem instances.

1.3 Overview of the Thesis

The remainder of this thesis is structured as follows.

Chapter 2 first reviews the existing approaches developed for fitness landscape analysis with the emphasis on discussing their limitations and drawbacks, highlighting why the development of an effective approach for characterising fitness landscapes is valuable. Second, the static and dynamic algorithm configuration problems are introduced with a review of existing approaches for both problems. This helps to identify the need and importance for incorporating fitness landscape analysis to build enhanced techniques for tackling both the static and dynamic algorithm configuration problems.

Chapter 3 presents the definition of the escape probability and a theoretical analysis on investigating the relationship between the escape probability and the expected runtime. Two further developments achieved based on the escape probability are then described. First, the fitness-probability cloud is proposed as an overall characterisation of fitness landscapes. Second, a problem difficulty measure, the accumulated escape probability is defined to explicitly quantify the problem difficulty with respect to algorithms. Finally, the sampling method used to estimate the accumulated escape probability in practice is introduced.

Chapter 4 first provides the reformulation of the static algorithm configuration as a decision problem. The generic (problem-independent) approach to perform per-instance algorithm configuration is described and illustrated through the case study of automatically configuring the $(\mu + \lambda)$ EAs for solving the unique input output sequence problem arising from software testing. In particular, an experimental study has been conducted to compare the proposed approach with the ParamILS which is a state-of-the-art algorithm configuration method. Furthermore, the approach to perform elementary landscape analysis is described with a case study on developing a novel and effective stochastic hill climbing algorithm for solving the next release problem (NRP) in software engineering.

Chapter 5 proposes to incorporate predictive information provided by the predictive problem difficulty measure to design a systematic mechanism for adaptive heuristic search. A generic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure is presented and applied to develop the fitness landscape based adaptive search algorithm (FAS) for the minimum vertex cover problem (MVC). A detailed experimental study is performed to evaluate the performance of FAS, and to further understand the behaviours of FAS and the predictive problem difficulty measure on instances with different characteristics.

Finally, Chapter 6 concludes this thesis by reviewing the main contributions of the preceding chapters and remarking on directions for further research.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

This chapter provides an introduction to the motivation behind fitness landscape analysis and a review of previous approaches. Despite many successful applications of metaheuristics in solving complex optimisation problems, there is in fact very limited understanding of which algorithms, or algorithm variants are best suited for solving which kinds of problems. Fitness landscape analysis is a powerful analytical tool that could be used to address this problem. In particular, two broad classes of approaches for characterising fitness landscapes and thus the problem difficulty with respect to metaheuristics, i.e., the qualitative approaches and the quantitative approaches, are reviewed and discussed. In addition to the two main classes of approaches, a promising but less general approach is introduced, which theoretically characterises a particular class of fitness landscapes, where the objective function is an eigenfunction of the Laplacian of the graph induced by the neighbourhood operator, as an elementary landscape.

Whilst fitness landscape analysis focuses on understanding problem difficulty in relation to algorithms in an analytical way, in practice, a number of approaches have been developed to directly tackle the problem of finding the best configuration of a given algorithm for solving a particular problem. This is motivated by the fact that metaheuristics are general algorithmic frameworks with two general classes of parameters to tune [86], namely, the behavioral parameters (mainly numerical parameters associated with search operators/heuristics) and the structural parameters (e.g., those related with the encoding and the choice of search operators/heuristics). Although some work refer to setting numerical parameters of metaheuristics as parameter tuning, this thesis is interested in the algorithm configuration, which concerns not only the problem of tuning numerical parameters, but of selecting and combining discrete building blocks (e.g., categorical parameters such as choice of search operator/heuristic) to build an effective algorithm.

In the literature, this algorithm configuration problem has been tackled under two distinct formulations. First, the static algorithm configuration, which takes place before actually running the algorithm and determines a priori the optimal configuration of the target algorithm for solving a particular problem. Second, the dynamic algorithm configuration, which is determining an optimal time-varying schedule for applying different algorithm configurations during the search process. Both static and dynamic algorithm configuration are extensively studied in the literature with a number of developed techniques. This chapter formally defines both the static and dynamic algorithm configuration problems, with the existing approaches reviewed and discussed.

This chapter identifies the fundamental link between fitness landscape analysis and algorithm configuration, in the sense that in-depth understanding of problem difficulty in relation to algorithms gained through fitness landscape analysis should naturally be of help in determining the optimal configuration of a given algorithm for solving a particular problem. Although it sounds very promising to incorporate the fitness landscape analysis in addressing the algorithm configuration problem, this chapter reviews the literature in the fields of fitness landscape analysis and algorithm configuration, and identifies three important research issues to be fully addressed:

- In fitness landscape analysis, there is a lack of a reliable and effective approach which can potentially be incorporated to build enhanced methods to directly address the algorithm configuration problem.
- In static algorithm configuration, most existing approaches either produce a one-size-fits-all configuration of an algorithm for an entire set of instances [66] or perform per-instance configuration for a particular problem by making use of problem-specific features. There is a lack of a generic (problem-independent) approach to automatically find the optimal configuration for a given algorithm on a per-instance base.
- In dynamic algorithm configuration, the optimal configuration of a heuristic algorithm is determined based on utilisation of historical information only under the assumption that a configuration that performed well in the past is bound to perform well in the future. There is a lack of a predictive method which can determine the optimal configuration dynamically based on the expected performance of candidate configurations instead of past performance only.

This chapter proceeds as follows. Section 2.1 reviews the previous approaches for characterising fitness landscapes and measuring problem difficulty with respect to algorithms, including the qualitative and the quantitative approaches, as well as elementary landscape. A discussion is also provided in this section to summarise the strengths and weaknesses of the existing approaches. Section 2.2 first formally defines the static algorithm configuration problem, then reviews and discusses a number of existing approaches. Section 2.3 describes the motivation to formulate the algorithm configuration as a dynamic problem, and formally defines the dynamic algorithm configuration problem. Again a number of previous approaches are reviewed and discussed. Finally, Section 2.4 presents a summary of this chapter, highlighting the need and importance of resolving the research issues identified in the literature review and the contributions to be made by the work presented in this thesis.

2.2 Existing Approaches for Fitness Landscape Analysis

Metaheuristics are approximate optimisation techniques which can find good solutions for hard optimisation problems within reasonable time but with no guarantee of finding the optimal solutions. There are a plethora of techniques within the field of metaheuristics such as Stochastic Local Search (SLS), Simulated Annealing (SA), Evolutionary Algorithm (EA), Ant Colony Optimization (ACO) and others. Also, since metaheuristics are general algorithmic frameworks, even a metaheuristic algorithm can have a number of variations under different configurations. Given such a plethora of algorithms and algorithm variants, for solving a given optimisation problem, the difficulty in selecting the best suited algorithm has been a long standing challenge for practitioners.

Furthermore, over the years the research focus in metaheuristics has largely been on the algorithmic side, where a large number of algorithms have been developed to solve one or more optimisation problems. In contrast, relatively little attention has been paid to studying the implications behind the empirical results, i.e., the analysis of which problems the proposed algorithm will perform well or poorly on and why. This lack of in-depth understanding of problem difficulty in relation to algorithms has made it more difficult to identify the best suited algorithm for solving a given problem. In the meantime, the necessity of addressing the problem of finding the best suited algorithm for solving a particular problem has been confirmed by the well-known no free lunch theorem [133] which states that no one optimisation algorithm is superior to the other on all problems, i.e. a single best algorithm for every problem does not exist.

Analysing characteristics of fitness landscapes, also referred to as the fitness landscape analysis, is a powerful analytical tool, particularly for understanding characteristics of optimisation problems and the associated behaviours of metaheuristics in optimising them. Therefore, the results obtained through fitness landscape analysis can be of help in understanding which algorithms, or algorithm variants are best suited for solving which kinds of problems, and thus in determining the best suited algorithm for solving a particular problem. The notion of fitness landscapes, originally proposed in [134], underlies a large body of work in problem difficulty studies in the field of metaheuristics. Typically a fitness landscape is defined under the notion of neighbourhood/distance, therefore the same fitness function can have many different fitness landscapes defined under different neighbourhood operators. Formally, the fitness landscape is defined as a triple (X, N, f), where X is a set of candidate solutions, the objective function $f : X \longmapsto \Re$ assigns a real-valued fitness to each solution in X and the neighbourhood operator $N : x \longmapsto N(x)$ imposes a neighbourhood structure among X. Given a candidate solution $x \in X, N(x)$ is the neighbourhood set that is obtained by applying one step of the neighbourhood operator.

Many observations can be made on the visualisation of such a fitness landscape (X, N, f) for properties such as ruggedness, smoothness, neutrality and etc, which can give an indication about the problem difficulty for a search algorithm. However, it is generally impossible to plot the fitness landscapes of most practical problems due to the huge size of the search space and the complexity of neighbourhood structure. Even assuming a fitness landscape can be plotted, the mere observation of its shape still lacks formality [51], it is then suggested that the ideal form of fitness landscape analysis would be to condense critical information on fitness landscapes into an algebraic measure [95].

Motivated by this distinction, a classification of the existing approaches for fitness landscape analysis is provided to break them into two main classes. First, the qualitative approaches which focussed on describing which characteristics make a fitness landscapes hard to optimise and which do not, based on the observations made on the visualisation of such a fitness landscape. On the other hand, quantitative approaches propose to condense the critical information on fitness landscapes into an algebraic measure to explicitly quantify problem difficulty with respect to search algorithms. In addition to these two main classes of approaches which focus on direct difficulty studies of fitness landscapes, a promising but less general approach theoretically characterises a particular class of fitness landscapes, where the objective function is an eigenfunction of the Laplacian of the graph induced by the neighbourhood operator, as elementary landscape.

2.2.1 Qualitative Approaches

Most early approaches emerging in the study of fitness landscapes attempted to link certain characteristics of fitness landscapes to problem difficulty, where characteristics such as isolation [44], multi-modality (presence of more than one local optimum) [63] and deception (presence of misleading information) [30, 38] were considered to be what make a problem hard for search heuristics. Many of these approaches consisted of constructing functions that should a priori be easy or hard for search algorithms to solve [95], e.g., isolation, multi-modality and deception. It is clear that an isolated fitness landscape (needle-in-a-haystack) is difficult for search heuristics. But in terms of multi-modality and deception, Naudts and Kallel [91] observed that fitness landscapes with those characteristics are not necessarily hard for search heuristics, and vice versa. Moreover, many counterexamples have been identified. Horn and Goldberg [63] constructed a multi-model function that is easy to solve. Also a unimodel long path problem [64] was proposed which can require exponential time to solve with hill climbing and GA. Wilson [132] constructed some deceptive functions that are GA-easy. Vose and Wright [122] presented a fully non-deceptive function which defeats the GA by an exponential number of sub-optimal attractors.

The fitness landscapes have also been characterised using properties of smoothness, ruggedness and neutrality [120], which attempt to link the geometric properties of fitness landscapes to the problem difficulty with respect to search heuristics. Intuitively a unimodal, very smooth and regular landscape seems straightforward for a searcher to locate the optimal solution. The opposite is true for a rugged fitness landscape with many local optima that are irregularly distributed, on which the local search algorithms would surely stagnate, even global search algorithms (as opposed to local search) might have trouble in optimising a rugged landscape. In general, search algorithms struggle to optimise very rugged fitness landscapes. Also, the neutral regions on fitness landscapes can make the search stagnate since the objective function is unable to provide any useful information when the search is on plateaus.

Recently the concept of fitness cloud [121] has been proposed to obtain a visual rendering of the evolvability properties of fitness landscapes. Let Γ be a set of individuals sampled from the search space and let $f_i = f(\gamma_i)$ where f is the fitness function. Then for each $\gamma_i \in \Gamma$ generate K neighbours by applying a genetic operator to γ_i and let f'_i denotes the maximum fitness value among K neighbours of γ_i. Finally the set of points $\{(f_1, f'_1), ..., (f_n, f'_n)\}$ is taken as the fitness cloud. Although the fitness cloud is predictive (does not require known global optima) and has proved its effectiveness in both Genetic Programming and GA, in practice it is vulnerable to

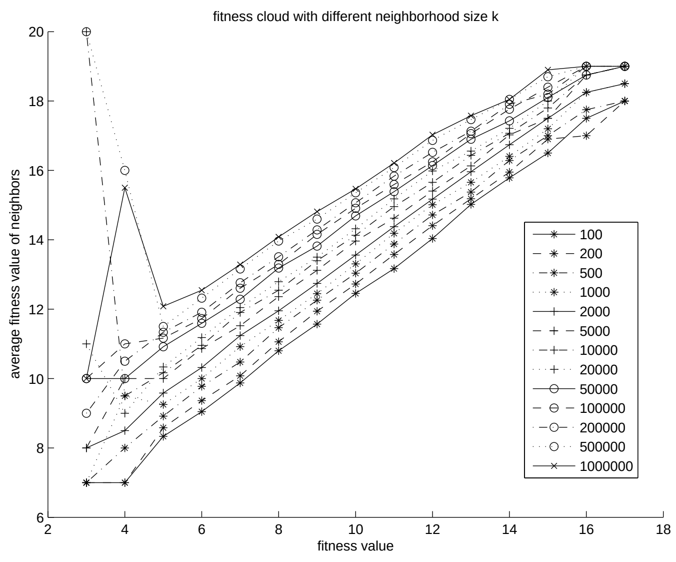


Figure 2.1: Plot of Fitness Cloud for OneMix function (Problem Size = 20) with varying Neighbourhood Sample Size K [82].

an experimental parameter which is the neighbourhood sample size K and there is no proper method in tuning this parameter [82]. As an illustration, for the OneMix function (problem size = 20) w.r.t. the bitwise mutation with probability 1/n, 13 different fitness clouds are generated as a result of using 13 different values of K varying from 100 to 10000000. The results showed that the fitness cloud is not a reliable characterisation of fitness landscapes unless the parameter K can be appropriately tuned.

The qualitative approaches for fitness landscape analysis summarised above can provide straightforward interpretations of the problem difficulty based on the observations made for certain characteristics of visualised fitness landscapes. However, it is generally impossible to visualise the fitness landscapes of most practical problems due to the huge size of the search space and the complexity of neighbourhood structure. Even assuming a fitness landscape can be plotted, the mere observation of its shape still lacks formality [51]. As for the fitness distribution [14] and information landscapes [15], they are not predictive since the information about the global optima is required.

2.2.2 Quantitative Approaches

The qualitative approaches for fitness landscape analysis only apply to low-dimensional problems where it is possible to plot the associated fitness landscapes, then observations can be made for the geometric properties of fitness landscapes such as ruggedness, smoothness, neutrality, etc. Nevertheless, fitness landscapes are unlikely to be plotted for most practical problems, due to the huge size of the search space as well as the complex neighbourhood structure. Even assuming a fitness landscape can be plotted, the mere observation of its shape still lacks formality [51], as a result, further to the qualitative characterisations of fitness landscapes, it is suggested to characterise fitness landscapes in a quantitative way, i.e., to compress the critical information on fitness landscapes into a single algebraic measure to explicitly quantify the problem difficulty with respect to algorithms. Typically such a problem difficulty measure is defined based on the fitness function and the underlying structure of the search space. Along this line of consideration, many problem difficulty measures have been proposed.

An early attempt in defining a useful problem difficulty measure is the epistasis variance [28], which assumes a binary representation and is computed based on the fitness function only. Basically the epistasis is measured under a linear composition of a string solution from its bits. The epistasis variance of the linear decomposition of the function is used to estimate the amount of non-linearity in the function. As a result, the epistasis variance gives a single value (from 0 to a non-normalised positive number), where 0 indicates no dependency between genes.

Weinberger proposed the autocorrelation function and correlation length [126] focussing on quantifying the geometric properties of fitness landscapes, i.e., the ruggedness. This approach is based on random walks through a binary fitness landscape. For all landscapes, for a sequence of fitness values obtained from random walk through the fitness landscape, calculate the correlation with the same sequence of values a small distance away. For autocorrelation function, the results are a plot of autocorrelation $q(s)$ against step size s (distance between sequences being correlated), where $q(s) = 1$ indicates maximal correlation and a value of $q(s)$ close to 0 indicates almost no correlation. The correlation length is a single numerical value computed based on the autocorrelation function, which indicates the distance beyond which the majority of points become uncorrelated, and a smaller value indicates a more rugged landscape. The main drawback of the correlation length as a problem difficulty measure lies in that it considers the problem difficulty as independent of the algorithm, however, it is widely acknowledged that the problem difficulty can vary for different search algorithms.

Later a significant contribution to this field was given by Jones through the introduction of a problem difficulty measure called fitness distance correlation [70] for measuring the GA-hardness. The fitness distance correlation (fdc) uses the joint variation of distances and fitness values [70]. Given a set $F = \{f_1, \ldots, f_n\}$ of n fitness values and a corresponding set $D = \{d_1, ..., d_n\}$ of the n distances away from the nearest global optimum, S_F and S_D are standard deviations of F and D. the correlation coefficient r is defined as:

$$r = \frac{C_{FD}}{S_F \cdot S_D}, where C_{FD} = \frac{1}{n}\sum_{i=1}^{n}(f_i - \overline{f})(d_i - \overline{d}). \tag{2.1}$$

Ideally we will have $r = 1$ when minimising (the fitness decreases while approaching a global optimum), and $r = -1$ when maximising (the fitness increases while approaching the global optimum).

One common limitation of most problem difficulty measures summarised above is they are not predictive, i.e., the optimal solutions to the problems are required to be known for computing these measures. This has also been confirmed in theoretical investigations by Jansen [67] which pointed out that the exact computation of these measures including fdc and epistasis variance requires the entire search space to be explored. Considering the huge size of the search space, one immediately realises that this is infeasible for most problems. In practice, statistical approaches such as sampling have to be employed, and only approximate values of these measures can be estimated based on samples of the search space. Motivated by the distinction between theoretical and empirical versions of problem difficulty measures, He et al. [54] proposed to classify realizations of difficulty measure into two types: namely exact and approximate realisations.

Nevertheless, both exact and approximate realisations of the existing problem difficulty measures are shown to be unreliable. In theory, Jansen [67] defined example functions where a simple evolutionary algorithm exhibits a completely different behaviour, whilst these functions are measured to be of the same difficulty in terms of fitness distance correlation and epistasis variance. It has also been proved that the expected runtime of this algorithm can vary from polynomial to exponential values even if the difficulty measure does not change at all. In practice, many counterexamples have been identified for fdc [3, 72, 100] and epistasis variance [102, 105] to show the approximate realisations of these problem difficulty measures are unreliable as well.

Furthermore, in an in-depth review of the previous problem difficulty measures, Reeves [103] pointed out their inherent flaws and concluded that a satisfactory problem difficulty measure with reference to metaheuristics is yet to be found. Given the disappointing outcome in finding a useful problem difficulty measure, Reeves [103] also raised an important theoretical question: can we know the difficulty of a problem without exploring the whole universe (search space)?

He et al. [54] provided a definite answer to this question. Assuming a worst-case perspective, it has been rigorously proved for both exact and approximate realisations, the predictive versions of the problem difficulty measure that can be computed in polynomial-time do not exist unless P = NP or BPP = NP. In other words, to find a useful predictive measure in general is impossible. This result is important in determining the right direction for the future developments of problem difficulty measures. Although it is impossible to find a reliable predictive measure in general, one can still design a predictive measure which is useful for a broad class of problem.

More recently, the negative slope coefficient (nsc) [119] has been proposed as a problem difficulty measure based on the notion of fitness cloud. Fitness cloud is a set of points $C = \{(f_1, f_1^{'}), \cdots, (f_n, f_n^{'})\}$, where f is the fitness function, f_i denotes the fitness value of a solution and $f_1^{'}$ denotes the fitness value of its neighbours. C can be partitioned into a number of discrete bins $C_1, \cdots, C_m$ such that $(f_a, f_a^{'}) \in C_j$ and $(f_b, f_b^{'}) \in C_k$ with $j < k$ implies $f_a < f_b$. For each of the line segments defined between the centroids of adjacent bins, a slope S_i is defined as:

$$S_i = \frac{f_{i+1}^{'} - f_i^{'}}{f_{i+1} - f_i} \tag{2.2}$$

Finally the negative slope coefficient is defined as:

$$nsc = \sum_{i=1}^{m-1} \min(0, S_i) \tag{2.3}$$

The hypothesis proposed in [119] stated that nsc should classify the problem difficulty in the following way: $nsc = 0$ indicates an easy problem and smaller values indicate more difficult problems when $nsc < 0$. nsc has been applied to a number of genetic programming (GP) problems and exhibited certain reliability. However, it has been identified that the partition of the abscissas of a fitness cloud can have a considerable impact on the performance of nsc. Limitations of the standard partitioning technique were shown in that it can generate bins which contain too few points. Furthermore, it was empirically shown that the nsc with this partitioning technique was unable to correctly predict the difficulty of two well known GP benchmarks: the multiplexer problem and the intertwined spirals problem [119]. In sum, *nsc* is unable to serve as a reliable problem difficulty measure unless the fitness cloud can be partitioned appropriately, and what is worse is that there is a lack of a formal partitioning method.

Independently, some problem difficulty measures which are theoretical in nature have been proposed. For example, the Kolmogorov complexity (KC), also known as algorithmic information theory, is a measure of an object related to the complexity of the computer program required to produce the object then halt. A discrete fitness function defined over a finite space can be described by a single binary string consisting of all possible output values of the function. The KC of this string is expected to capture the difficulty of the function [16]. Although the approach of using KC to quantify function complexity has been used extensively in theoretical studies and proofs, the KC, just like other measures which are theoretical in nature, cannot be practically implemented.

2.2.3 Elementary Landscape

The approaches for fitness landscape analysis summarised above focussed on gaining general understanding of problem difficulty in relation to algorithms. The emergence of elementary landscape provides a promising but less general approach, which characterises fitness landscapes by theoretically defining a particular class of fitness landscapes. The elementary landscapes have unique properties which can potentially be used to facilitate the search.

First observed by Grover [49] that the landscapes of certain combinatorial optimisation problems such as Travelling Salesman Problem (TSP), could be characterised by a wave equation. Stadler [109] formally defined this kind of landscape as "elementary landscapes".

For a fitness landscape (X, N, f), where X denotes a set of candidate solutions, the objective function $f : X \longmapsto \Re$ assigns a real-valued fitness to each solution in X and the neighbourhood operator $N : x \longmapsto N(x)$ imposes a neighbourhood structure among X. Let $G(X, E)$ be the underlying graph induced by N and assume G is regular with vertices of degree d. $A \in \mathbb{R}^{|X|\times|X|}$ is the adjacency matrix of G, if x_1 and x_2 are adjacent, $A(x_1, x_2) = 1$. $\triangle = A - dI \in \mathbb{R}^{|X|\times|X|}$ is the Laplacian of G.

On an arbitrary fitness landscape, f and N are unrelated. However, on an elementary landscape, the following wave equation holds:

$$\triangle f = \lambda f \tag{2.4}$$

λ is a scalar. In other words, a landscape is elementary when the objective function is an eigenfunction of the Laplacian of the graph induced by the neighbourhood operator.

Elementary landscapes possess unique properties which are considered beneficial for the search. In general, the properties are classified into two main categories. Implicit properties, landscapes with this property tend to be relatively smooth when contrasted to other combinatorial optimisation problems with well-studied local move operators, which could be considered to be an advantage for local search algorithms [130]. And the wave equation also imposes constraints on the structure of local optima and precludes the existence of certain plateau structures. Explicit properties, a wave equation in terms of the expected value of the neighbours is proposed, which more concretely expresses the properties of elementary landscapes [130]. Suppose x is some fixed but arbitrary element of X, y is an element drawn uniformly at random from the neighbourhood set $N(x)$ of x and $\overline{f}$ is the mean value over all solutions in X. On an elementary landscape, the following wave equation holds.

$$E[f(y)] = f(x) + \frac{k}{d}(\overline{f} - f(x))$$

for k which is fixed for the entire landscape. Since y is drawn uniformly at random, the expected value of the fitness value of a neighbour y is always equal to the average fitness value over all solutions in the neighbourhood [130]. In addition to computing the expected fitness value of the full neighbourhood can be predicted by the wave equation. It is even possible to expand a partial neighbourhood during the search, and predict for the remaining neighbourhood. This property gives significant insight to the search algorithm that could be explicitly applied in designing algorithms. The wave equation also imposes constraints on the structure of local optima and precludes the existence of certain plateau structures. One of the following observations by Codenotti and Margara [23] is true.

- if $f(x) = \overline{f} \quad f(x) = E[f(y)] = \overline{f}$
- if $f(x) < \overline{f} \quad f(x) < E[f(y)] < \overline{f}$
- if $f(x) > \overline{f} \quad f(x) > E[f(y)] > \overline{f}$

Grover [49] observed similar consequences. Let Z_{min} and Z_{max} be a local minimum and a local maximum, respectively. Then

$$Z_{min} < \overline{f} < Z_{max}$$

In other words, all local minima lie below the average function value of the search space.

Barnes et al. [9] classified the elementary landscapes as either smooth or rugged. The wave equation holding for smooth elementary landscapes has resulted in several properties, which include relative smoothness, constraints on certain plateau structures and local optima, as well as allowing for predictions about the fitness values of partial or full neighbourhoods during search, etc. In addition, arbitrary fitness landscapes can be decomposed into a superposition of elementary landscapes.

Whitley et al. [130] also proved that for a plateau P on a (non-flat) elementary landscape, if $x \in P$ has only equal and not improving neighbours, then there cannot exist a solution $z \in P$ with only equal and improving neighbours. A plateau is a set P of candidate solutions in X such that for all $a, b \in P$, $f(a) = f(b)$ and there is a path $(a = x_1, x_2, ..., x_k = b)$ such that $x_{i+1} \in N(x_i)$. Plateaus (also known as neutral networks) are structural features that arise in many combinatorial problems [39]. Plateau structure is a challenge for local search that can cause the algorithm to cease progress.

Whitley et al. [130] observed that several interesting consequences that arise from the expected value equation. In most practical elementary landscapes studied, the objective function for a particular candidate solution can be written as a linear combination of a subset of a collection of components. A good example is TSP, in which a fitness function is a linear combination of edge weights. Let C be a set of real valued components and there exists $C_x \subset C$ such that $f(x) = \sum_{c \in C_x} c$. The set C_x is referred to as the intracomponents of a solution x and the set $C - C_x$ as the intercomponents of x. When a local search move has been made from an incumbent solution x to a neighbouring solution, an exchange of components is made. In particular, a subset of the intracomponents is removed and a subset of the intercomponents is added.

On this basis, Whitley et al. [130] constructed a component-based model that can be used to characterise a neighbourhood structure. In this model, the neighbourhood size is regular and denoted by d. The model consists of the following equations.

$$\overline{f} = p3 \sum_{c \in C} c$$

$$E\{f(y)\} = f(x) - p1f(x) + p2(\sum_{c \in C} c - f(x))$$
$$= f(x) - p1f(x) + p2(\frac{1}{p3}\overline{f} - f(x))$$

where $0 < p1 < 1$ is the proportion of the intracomponents that are removed from the solution in one move, $0 < p2 < 1$ is the proportion of the intercomponents that are added to the solution in a move. Finally, $0 < p3 < 1$ is the proportion of the total components in C that contribute to the cost function for any randomly chosen solution, which is independent of the neighbourhood size. Based on this model, a component theorem has been proposed to determine whether a landscape is elementary or not:

Theorem 1 *(Component Theorem [129])*

If $p1, p2$ and $p3$ (must be constants) can be defined for any regular landscape such that the evaluation function can be decomposed into components where $p1 = \alpha/d$ and $p2 = \beta/d$ and

$$\overline{f} = p3 \sum_{c \in C} c = \frac{\beta}{\alpha + \beta} \sum_{c \in C} c$$

then the landscape is elementary.

It has been shown that the elementary landscape is a special type of landscape with unique properties which have implications in both theory and practice. However, an arbitrary fitness landscape is not always elementary, actually only a small number of combinatorial optimisation problems have search spaces that correspond to elementary landscapes [22].

In fact, arbitrary fitness landscapes can be decomposed into a superposition of "elementary landscapes" via a Fourier series expansion. A series expansion $f(x) = \sum_{i=1}^{N} \alpha_i \varphi_i(x)$, where φ_i forms a complete and orthonormal system of eigenfunctions of the graph Laplacian, is termed a Fourier series expansion of the objective function. This decomposition is helpful in a sense that some statistical properties of the landscape could be computed and the decomposed elementary landscapes can be studied individually. The information about the effective hardness of an elementary landscape is contained in the relative ordering of the associated eigenvalues [110]

More recently, Chicano et al. [22] proposed a methodology to addresses this issue to generalise the definition of elementary landscape to an arbitrary landscape. They presented an algebraic method that can be used to decompose the fitness function of an arbitrary combinatorial optimisation problem as a superposition of multiple elementary landscapes if the underlying neighbourhood is symmetric. In addition to generalise the definition of elementary landscape to an arbitrary landscape, such a decomposition has more uses in both theory and practice. In theory, the landscape decomposition of a problem can be used to compute the exact expression for the autocorrelation functions, the autocorrelation coefficient, and the autocorrelation length [126]. In practice, the decomposition allows the search to obtain the average fitness value of the neighbourhood of any solution without expanding and evaluating all the solutions in the neighbourhood.

2.2.4 Discussion

Despite extensive research on fitness landscape analysis and a number of techniques developed over the years, very few techniques are used in practice. This is partly because fitness landscape analysis being complex in itself, which has made it inaccessible for many practitioners. On the other hand, a considerable amount of computational overhead will be incurred by fitness landscape analysis, however, the results given by the existing approaches are still unreliable.

As summarised above, the qualitative approaches for fitness landscape analysis provided a straightforward indication of the problem difficulty with respect to search algorithms. However, these qualitative approaches only applied to low-dimensional problems such as artificially constructed functions with simple structure, and they were unable to deal with most practical problems with huge search space and complex neighbourhood structure. Even assuming a complex fitness landscape can be visualised, the mere observation of its shape still lacks formality, not to mention the many counterexamples identified which showed that the geometric properties of fitness landscapes are neither a sufficient nor a necessary condition for the problem difficulty with respect to algorithms. From the perspective of practitioners, the mere observation of fitness landscapes cannot be used to compare and discriminate the expected performance of different algorithms for solving a given problem, and therefore are of limited use in addressing the problem of finding the best suited algorithm for solving a given problem.

The existing quantitative approaches were either shown to be unreliable by many counterexamples or unable to be used in practice due to their own limitations, e.g., require known global optima, vulnerable to experimental parameters. It is worth noting that finding a useful problem difficulty measure in general is impossible. The emphasis is then on developing a useful problem difficulty measure for a broad class of problems.

Given a lack of a reliable approach for fitness landscape analysis, further studies are required to develop a reliable approach for characterising fitness landscapes and measuring problem difficulty with respect to search algorithms, ideally for a broad class of problems. In the interest of practical applications, it is interesting to not only develop a standalone method for fitness landscape analysis, but to explore the possibility of incorporating such a fitness landscape analysis method with other powerful techniques such as the machine learning algorithms to more directly tackle the problem of finding the best suited algorithm for solving a given problem.

2.3 Static Algorithm Configuration

Metaheuristics are general algorithmic frameworks, even a metaheuristic algorithm can have many variants if different configurations are used. It is widely acknowledged that finding good algorithm configurations is essential to obtain robust and high algorithm performance. For example, a simulated annealing algorithm can perform extremely differently with different values of the temperature parameter even on the same problem instance. Furthermore, it has been observed that an algorithm requires different configurations in order to find good solutions for different problem instances [41, 94, 137].

As of yet, finding the best algorithm configuration for solving a particular problem instance remains one of the persisting challenges in the field of metaheuristics. This is due to the fact that there is in fact very little understanding of which algorithm configurations are best suited for solving which problem instances. Previous work revealed that a large fraction of time in algorithm development has been spent on configuring the algorithm [1, 11], since in many cases the algorithm is configured manually by trial and error, and the configuration space is essentially very large even for a handful of parameters [79]. Clearly it is not only inefficient and laborious, but very unlikely to locate the optimal algorithm configuration, if the algorithm were to be configured manually. Therefore, developing automatic algorithm configuration methods is of high practical relevance in several aspects [66]:

- **Algorithm Development** Manually configuring an algorithm is both time-consuming and labour-intensive. The use of automatic algorithm configuration techniques can lead to significant reduction of the time spent on configuring the algorithm, one can them focus on the algorithm design and testing, which will potentially achieve better results than algorithms developed and configured with manual methods.

- **Empirical Comparison of Algorithms** When comparing different metaheuristics for solving an optimisation problem, it is difficult to distinguish whether an algorithm outperforms another since it is fundamentally better suited or it is more appropriately configured. Automatic algorithm configuration methods can alleviate this problem by means of performing fair configuration for different algorithms, and thus facilitate objective comparison between algorithms.

- **Real-world Applications** In practical applications of metaheuristics, the performance of an algorithm on real-world problem instances often depends critically on the use of appropriate configurations of the algorithm. In the meantime, finding appropriate configurations requires extensive knowledge about both the problem and the algorithm, but the practitioners often have limited knowledge about the algorithm, and the performance of the algorithm simply using the default configuration optimised on benchmarks might significantly deteriorate on real-world instances. Automatic algorithm configuration methods can provide a systematic solution to this problem.

This section first formally defines the static algorithm configuration problem, then reviews and discusses the existing approaches for static algorithm configuration.

2.3.1 Static Algorithm Configuration Problem

To avoid potential confusions between the algorithm to be configured and the method developed to undertake the configuration task, the former is referred to as the target algorithm and the latter as algorithm configuration method [66]. Let $\mathcal{A}$ denote the target algorithm with a set of parameters P which can be numerical, ordinal (large, medium, small) or categorical (choice of search operator/heuristic). The configuration space Θ consists of the domain of values for each parameter $p \in P$. Let $\theta = \{p_1 \cdots p_k\}$ denote a configuration of the target algorithm $\mathcal{A}$, $\mathcal{H}$ is a function to measure the performance of the target algorithm executed under configuration θ on a problem instance. The evaluation function $\mathcal{H}$ can be in many different forms, for example, one might be interested in minimising the runtime of the target algorithm for finding a solution or maximising the quality of the solution found within fixed time. The overall goal of an algorithm configuration method is to find a configuration θ of the target algorithm $\mathcal{A}$ for each problem instance $I \in \mathcal{D}$, where the performance of the target algorithm $\mathcal{A}$ over a set of instances $\mathcal{D}$ is optimised. The static algorithm configuration problem is formally defined as:

Definition 1 *(Static Instance-based Algorithm Configuration Problem).*

An instance of the algorithm configuration problem is a tuple $\langle \mathcal{A}, P, \Theta, \mathcal{D}, \mathcal{H}, m \rangle$, *where:*

- $\mathcal{A}$*: A parameterised algorithm;*
- P*: Parameters of* $\mathcal{A}$*;*
- Θ*: Configuration space of* P*;*
- $\mathcal{D}$*: A class of problem instances;*
- $\mathcal{H}$*: A function that measures the performance of running* $\mathcal{A}(\theta)$*,* $\theta \in \Theta$ *on an instance;*
- m *is a statistical parameter. (Examples are expectation, mean, and variance.)*

The cost function for any candidate configuration θ *is:*

$$c(\theta) = m_{I \in D}[\mathcal{H}(\mathcal{A}, \theta, I)] \tag{2.5}$$

Assuming minimising $\mathcal{H}$, the static algorithm configuration problem is to find a configuration $\theta \in \Theta$ of $\mathcal{A}$ for each problem instance $I \in \mathcal{D}$ such that $c(\theta) = m_{I \in D}[\mathcal{H}(\mathcal{A}, \theta, I)]$ is minimised.

2.3.2 Automatic Algorithm Configuration Methods

One of the main challenges in developing effective automatic algorithm configuration methods lies in determining whether there are patterns or rules governing the choice of algorithm configurations, and whether such patterns can be learnt [79]. In the last two decades, many approaches have been developed to determine a priori the best suited configuration of the target algorithm for solving a problem. In general, the existing automatic algorithm configuration methods can be classified into two categories:

- **Generic, one-size-fits-all Configuration Methods** Generic (problem-independent) methods which produce a one-size-fits-all configuration of a given algorithm for an entire set of problem instances.
- **Problem-dependent, per-instance Configuration Methods** Problem-dependent methods which perform algorithm configuration on a per-instance base. Each method only applies to one particular optimisation problem by making use of problem-specific features.

2.3.2.1 Generic, one-size-fits-all Configuration Methods

Gratch and Dejong [47] developed a hill climbing algorithm to directly search the configuration space by taking moves when a neighbouring configuration is statistically significantly better than the incumbent configuration. This hill climbing algorithm was successfully applied to configure a scheduling algorithm with five parameters.

MULTI-TAC [89] takes as input a number of generic heuristics and a set of problem instances, then adapts the generic heuristics to the problem domain and automatically generated problem-specific LISP programs as their implementations. A beam search is then performed to select the best LISP implementation where each program is evaluated over an entire set of problem instances.

Adenso-Diaz and Laguna [1] developed an automatic algorithm configuration system called CALIBRA, based on a combination of experimental design and gradient descent. CALIBRA uses a fixed training set for evaluation. The experimental results showed great promise in that CALIBRA was able to find parameter settings for six independent algorithms that matched or outperformed the respective originally proposed parameter configurations. CALIBRA's main drawback is its limitation to tuning numerical and ordinal parameters, and to tuning a maximum of five free parameters [66].

Birattari et al. [12] proposed F-race based on adaptations of racing algorithms in machine learning to the algorithm configuration problem, and applied F-race to the configuration of stochastic local search algorithms. Various applications of F-race have demonstrated good performance. However, since at the start of the procedure all candidate configurations are evaluated, this approach is limited to situations in which the number of candidate configurations considered simultaneously is not too large.

More recently, Hutter et al. proposed ParamILS [66] to employ the iterative local search for finding the optimal algorithm configuration of a given algorithm for solving a set of problem instances. ParamILS has a large number of academic and industrial applications, which has yielded substantial improvements of heuristic algorithms for hard combinatorial problems, such as propositional satisfiability (SAT), mixed integer programming (MIP), AI planning, answer set programming (ASP), and timetabling. ParamILS is a direct search method which searches the configuration space using an iterative local search method (ILS). It uses a combination of default and random settings for initialization, employs iterative first improvement as a subsidiary local search procedure, uses a fixed number (s) of random moves for perturbation, and always accepts better or equally-good configurations, but re-initializes the search at random with a small probability. Furthermore, it is based on a one-exchange neighbourhood, that is, it always considers changing only one parameter at a time. ParamILS returns a one-size-fits-all configuration for an entire set of problem instances.

The approaches summarised above have in common one drawback that they only produce a one-size-fits-all configuration of the target algorithm for an entire set of problem instances. However, given a search algorithm for solving an optimisation problem, it has been observed that the algorithm requires different configurations in order to find good solutions for different problem instances [137].

2.3.2.2 Problem-dependent, per-instance Configuration Methods

Patterson and Kautz [94] first introduced the approach to perform automatic algorithm configuration on a per-instance base. This approach was specifically designed to configure local search heuristics for solving SAT problems.

Hutter and Hamadi [65] proposed an instance-aware algorithm configuration method based on approaches which can predict performance for the problem instance at hand and each (continuous) parameter configuration. The proposed algorithm then simply chooses the configuration that minimises the prediction. The experiments demonstrated that this approach can fairly accurately predict the run-time of SAPS, one of the best-performing stochastic local search algorithms for SAT. The application of this method is limited to SAT problems since the performance prediction method is based on SAT problems.

Cavazos and O'Boyle [21] proposed automatically selecting the best suited configuration of optimisation heuristics on a per method basis within a dynamic compiler. This approach uses the machine learning technique of logistic regression to automatically derive a predictive model that determines which configuration of optimisation heuristics to apply based on the features of a method. The application domain of this method is limited to method optimisation in compiling.

Xu et al. proposed SATzilla [137] which constructs per-instance algorithm portfolios for SAT. SATzilla uses 48 features, most of which are SAT-specific features.

The approaches summarised above aim to automatically choose the optimal configuration of the target algorithm on a per-instance basis. The main drawback of these approaches is in that each method can only be used for a particular problem due to the use of problem-specific features.

2.4 Dynamic Algorithm Configuration

Traditionally, algorithm configuration methods determine a priori the most appropriate configuration of a given heuristic algorithm for solving a particular problem instance. However, there is both empirical and theoretical evidence showing that the most effective configuration of a given algorithm for solving a particular problem instance can vary during the search process [116]. For example, theoretical analysis of mutation operators on binary encoded problems concluded that the mutation probability should be decreased as the genetic algorithm is approaching the global optimum [90]. Empirically Davis [29] used a time-varying schedule of operator probabilities and observed improved performance.

Therefore, the performance of a heuristic algorithm depends not only on the initial choice of its configuration, but strongly on the management of its configuration along the search. This motivates the development of dynamic algorithm configuration methods, often referred to as adaptive heuristic algorithms, which dynamically adapt their configurations (search operators, numerical parameters, etc.) during the search process. The behaviours of such heuristic algorithms are adapted to specific characteristics of problem instances. These algorithms have gained increasing popularity in recent years, as such algorithms with improved generality would increase the potential applicability for solving a broad class of problems with different characteristics.

Adaptive heuristic algorithms are extensively studied in the literature with a number of approaches developed. The design of adaptive heuristic algorithms mainly concerns selecting the most effective configuration for the algorithm at each decision point during the search procedure. Many such mechanisms to control the adaptation of algorithm configurations have been proposed, which fall within three main categories [35]: Deterministic, algorithm configurations are predefined functions of time; Self-Adaptive, algorithm configurations are part of the genotype and optimized by evolution itself; Adaptive Rules: algorithm configurations are predefined functions of the search history.

This section first formally defines the dynamic algorithm configuration problem, then reviews and discusses a number of previous adaptive heuristic algorithms.

2.4.1 Dynamic Algorithm Configuration Problem

Let $\mathcal{A}$ denote the target algorithm with a set of parameters P which can be numerical, ordinal (large, medium, small) or categorical (choice of search operator/heuristic). The configuration space Θ consists of a set of pre-defined configurations for $\mathcal{A}$. Instead of determining a priori the most effective configuration, dynamic algorithm configuration changes the configuration of $\mathcal{A}$ at each decision point during the search process. Let the vector $\theta = \{\theta_1 \cdots \theta_k\}$ denote a sequence of configurations to use in one execution of the target algorithm $\mathcal{A}$, where $\theta_i \in \Theta$ is the configuration to select at decision point i. $\mathcal{H}$ is a function which measures the performance of the target algorithm executed under sequence θ on an instance I. The evaluation function $\mathcal{H}$ can be in many different forms, for example, we might be interested in minimising the runtime of the target algorithm for finding a solution or maximising the quality of the solution found within fixed time. The overall goal is to find a sequence of configurations θ of the target algorithm $\mathcal{A}$ for each problem instance $I \in \mathcal{D}$, where the performance of the target algorithm $\mathcal{A}$ over a set of instance $\mathcal{D}$ is optimised. The dynamic algorithm configuration problem is formally defined as:

Definition 2 *(Dynamic Algorithm Configuration Problem).*

An instance of the algorithm configuration problem is a tuple $\langle \mathcal{A}, P, \Theta, \theta, \mathcal{D}, \mathcal{H}, m \rangle$, where:

- *$\mathcal{A}$: A parameterised algorithm;*
- *P: Parameters of $\mathcal{A}$;*
- *Θ: Configuration space of P;*
- *$\theta = \{\theta_1 \cdots \theta_k\}$, $\theta_i \in \Theta$: A sequence of configurations.*
- *$\mathcal{D}$: A class of problem instances;*
- *$\mathcal{H}$: A function that measures the performance of running $\mathcal{A}(\theta)$ on an instance;*
- *m is a statistical parameter. (Examples are expectation, mean, and variance.)*

The cost function for a sequence θ is:

$$c(\theta) = m_{I \in D}[\mathcal{H}(\mathcal{A}, \theta, I)] \tag{2.6}$$

Assuming minimising $\mathcal{H}$, the dynamic algorithm configuration problem is to find a sequence of configurations θ of the target algorithm $\mathcal{A}$ for each problem instance $I \in \mathcal{D}$ such that $c(\theta) = m_{I \in D}[\mathcal{H}(\mathcal{A}, \theta, I)]$ is minimised.

2.4.2 Adaptive Heuristic Algorithms

Adaptive heuristic algorithms aim to select the most effective configuration of a given heuristic algorithm to use at each decision point during the search process. Many different mechanisms to control the adaptation of algorithm configurations have been developed, which fall within three main categories [35]:

- Deterministic: algorithm configurations are predefined functions of time.
- Self-Adaptive: algorithm configurations are part of the genotype and optimized by evolution itself.
- Adaptive Rules: algorithm configurations are predefined functions of the search history.

Deterministic methods essentially raise even higher difficulties than static algorithm configuration: as the optimal configuration of the target algorithm changes with time, these functions must pre-define a schedule for applying different configurations along the search process. Self-Adaptive method is acknowledged as one of the most effective approaches to evolutionary parameter setting, specifically in the framework of continuous parameter optimization [31]. In the general case however, self-adaptive approaches often significantly increase the size of the search space, and thus the complexity of the optimization problem (not only should a successful individual have good genes; it should also bear parameter values enforcing some effective transmission of its genes) [35].

Adaptive rules, also referred to as feedback-based control, use information from the search history to adapt the configuration of a given heuristic algorithm while solving the problem. In particular, adaptive rules aim at defining an online strategy to determine the most effective configuration on the fly. In the context of adaptive rules, the design of adaptive heuristic algorithms concerns two main issues, the credit assignment mechanism and selection mechanism, where the former assigns a reward to a configuration based on evaluating the contribution of the configuration to the overall performance and the later serves as a selection rule in charge of selecting the configuration to use.

2.4.2.1 Credit Assignment Mechanisms

Several mechanisms for credit assignment have been proposed which mostly concern how to compute the rewards to be assigned to candidate configurations. Most existing mechanisms mainly differ in the metric used for evaluating performance of candidate configurations.

Most approaches defined the performance metric as the fitness improvement between the offspring and the parents or other objects. To be more specific, the fitness improvement is assessed in comparison with i) the current best individual [29]; ii) the median fitness [71]; or iii) the parent fitness [116]. To avoid premature convergence of the population-based algorithms, Maturana and Saubion [87] took into account the population diversity and proposed a measure defined as a weighted sum of both the fitness improvement and the offspring diversity.

Instead of considering instantaneous or average improvement, Whitacre et al. [128] considered extreme improvements, using a statistical measure aimed at outlier detection in numerical optimisation. The experimental results showed that the proposed measure significantly outperformed its competitors on a set of continuous benchmark problems.

In addition, some other work proposed that the impact of candidate configurations on the overall performance should be measured after the genealogy of the outstanding offspring, e.g., rewarding the operators producing the ancestors of a good offspring according to a bucket brigade algorithm [29].

One common shortcoming of the credit assignment mechanisms summarised above is the rewards are determined based on the historical performance of candidate configurations only, however, it is noted that considering only the past performance can be misleading on problems with complex structure such as deception [30, 88]. Furthermore, the optimal reward assigned to a configuration is a dynamic random variable and the underlying distribution of this random variable changes as the search proceeds [29, 90].

2.4.2.2 Selection Mechanisms

Extensive research efforts have been devoted to developing effective selection mechanisms in the last two decades. Most selection mechanisms transform the rewards assigned to candidate configurations into a probability distribution consists of probabilities indicating the likelihoods for selecting the candidate configurations along the search process. Many different approaches are proposed for learning the optimal probability values of applying a fixed set of algorithm configurations.

Most existing methods belong to the probability matching type [24, 45, 59, 116, 127]. The basic probability matching selection rule computes each configuration's selection probability as the proportion of the configuration's reward to the total sum of all rewards. The main drawback of probability matching is that this can lead to the loss of some candidate configurations. If the selection probability of an algorithm configuration would become too low at some point, it would never be used again and its reward can no longer be updated. This is an unwanted property since the operator might become valuable again in a future stage of the search process [114]. To ensure no candidate configuration gets lost, a minimum selection probability can be enforced. In practice, all mildly relevant operators keep being selected, hindering the probability matching performance [35]. To address this issue, the adaptive pursuit method [114, 115] has been proposed, in which the selection probability is updated in such a way that the algorithm pursues the configuration that currently has the maximal reward. To achieve this, the pursuit method increases the selection probability of the configuration with the maximal reward and decreases selection probabilities of all other configurations.

Alternatively, there has been a class of methods based on the Multi-armed Bandit Paradigm [6, 27], which formulates the configuration selection as a Exploration vs. Exploitation (EvE) dilemma, where Exploitation aims at selecting the best rewarded configuration in the last stages of search whereas Exploration is concerned with checking whether other configurations might in fact become the best ones at some later stages. The EvE dilemma has been intensively studied in Game Theory, more specifically in the so-called Multi-Armed Bandit (MAB) framework [6]. The MAB framework considers a set of K independent arms, each one of which having some unknown probability of getting a (boolean) reward. The optimal selection strategy is one maximizing the cumulative reward along time. A vital limitation of the standard MAB framework lies in that it only considers a static environment (the unknown reward probability of any arm being fixed along time), whereas the adaptive algorithm is intrinsically dynamic (the quality of any configuration is bound to vary along evolution). Even though every configuration keeps being selected and it can ultimately be realised that some new configuration has become the best one, in practice this would need to wait way too long before the new best configuration can be discovered.

2.5 Summary

In summary, fitness landscape analysis, a powerful analytical tool in understanding the problem difficulty in relation to algorithms, shows great promise in overcoming a long standing challenge in the field of metaheuristics, i.e., finding the best suited configuration of a given algorithm for solving a particular problem instance, also referred to as the algorithm configuration problem.

Two main classes of approaches for fitness landscape analysis, as well as a promising but less general approach, have been reviewed. The qualitative approaches attempted to characterise the problem difficulty by means of describing which characteristics make the fitness landscape hard to optimise and which do not. However, many counter-examples have been identified to show that these characteristics are neither sufficient nor necessary for a difficult problem with respect to algorithms [63, 64, 91, 122]. Instead, the quantitative approaches proposed to condense the critical information on fitness landscapes to a single algebraic measure to explicitly quantify the problem difficulty in relation to algorithms. Further to the many counter-examples identified for previous problem difficulty measures [3, 67, 100, 102, 105], theoretically it has been concluded that a useful predictive problem difficulty measure can never be found [54]. As for the elementary landscape, where the objective function is an eigenfunction of the Laplacian of the graph induced by the neighbourhood operator, it is noted that only a small number of combinatorial optimisation problems have search spaces that correspond to elementary landscapes [22]. As a result, the applicability of this approach is significantly limited. In sum, despite extensive research on fitness landscape analysis and a large number of developed approaches, very few approaches are used in practice. This is due to the fact that most existing approaches are either proved to be unreliable or unable to be used in practice due to their own limitations (e.g. require known global optima).

The algorithm configuration problem has been studied extensively, under two distinct formulations. The static algorithm configuration problem is of determining a priori the optimal configuration of the algorithm for solving a particular problem, before applying the algorithm. In contrast, the dynamic algorithm configuration problem is of determining an optimal time-varying schedule for applying different algorithm configurations during the search process.

This chapter identified the fundamental link between fitness landscape analysis and algorithm configuration, in the sense that in-depth understanding of problems in relation to algorithms gained through fitness landscape analysis should naturally be of help in determining the optimal configuration of a given algorithm for solving a particular problem. Although fitness landscape analysis shows great promise in addressing the algorithm configuration problem, in order to effectively incorporate fitness landscape analysis to build enhanced techniques for algorithm configurations, three important research issues need to be fully addressed:

- In fitness landscape analysis, there is a lack of a reliable and effective approach which can potentially be incorporated to build enhanced methods to directly address the algorithm configuration problem.
- In static algorithm configuration, most existing approaches either produce a one-size-fits-all configuration of an algorithm for an entire set of instances [66] or perform per-instance configuration for a particular problem by making use of problem-specific features. There is a lack of a generic (problem-independent) approach to automatically find the optimal configuration for a given algorithm on a per-instance basis.
- In dynamic algorithm configuration, the optimal configuration of a heuristic algorithm is determined based on utilisation of historical information only under the assumption that a configuration performed well in the past is bound to perform well in the future. There is a lack of a predictive method which can determine the optimal configuration dynamically based on the expected performance of candidate configurations instead of past performance only.

This thesis aims to contribute towards addressing these three research issues. First, to develop a novel, effective approach for characterising fitness landscapes and measuring problem difficulty with respect to algorithms. Second, to incorporate fitness landscape analysis in building a generic approach which can perform automatic algorithm configuration on a per-instance base, and potentially to design novel algorithm configurations using the insight gained through fitness landscape analysis. Third, to incorporate fitness landscape analysis to build a predictive method which can determine the optimal configuration dynamically based on the expected performance of candidate configurations instead of the past performance only.

Chapter 3

FITNESS LANDSCAPE ANALYSIS FOR MEASURING PROBLEM DIFFICULTY WITH RESPECT TO METAHEURISTICS

3.1 Introduction

Despite many successes of metaheuristics in solving complex optimisation problems and a large number of developed techniques, relatively little attention has been paid to study the implications behind the empirical studies of the proposed algorithms. In fact, there is very limited understanding of which algorithms, or which algorithm variants are best suited for solving which kinds of problems.

Given this lack of understanding of problem difficulty with respect to algorithms, the answer to fundamental questions in the field of metaheuristics such as "how to determine if a problem is difficult or easy for a given search algorithm?", was sought after for over two decades, but no satisfactory answer has been found yet. In particular, the notion of fitness landscapes, originally proposed in [134], underlies a large body of work in problem difficulty studies with respect to metaheuristics. Most existing approaches for fitness landscape analysis proceed along two main routes. The qualitative approaches focus on describing which characteristics make the fitness landscape hard to optimise and which do not. However, it is generally impossible to plot fitness landscapes for most practical problems due to the huge size of search space as well as the complex neighbourhood structure. Even assuming a fitness landscape can be plotted, the mere observation of its shape still lacks formality [51]. Moreover, many counter-examples have been identified to show that these characteristics are neither sufficient nor necessary for a problem to be difficult [63, 64, 91, 122, 132]. Instead, the quantitative approaches propose to condense the critical information on fitness landscapes to a single algebraic measure to explicitly quantify the problem difficulty with respect to algorithms. However, the previous measures haven been shown to be unreliable both theoretically [67] and empirically [3, 72, 100, 102, 105]. More importantly, He et al. [54] have rigorously proved for both exact and approximate realisations, a reliable problem difficulty measure that can be computed in polynomial-time does not exist unless P = NP or BPP = NP. The emphasis is then not on finding a useful measure in general, but rather on developing a measure that can be used for a broad class of problems.

It is noted that the problem difficulty in relation to algorithms have been extensively studied in time complexity theory of metaheuristics and there exists a large number of results on convergence [46], runtime analysis [68], etc. Although theoretical analysis can provide useful information to better understand problem difficulty in relation to algorithms, the results from theoretical analysis are often under several assumptions that can hardly be satisfied in practical scenarios. Effective approaches are therefore required on how to make use of the theoretical results in practice.

This chapter proposes for the first time to bridge the gap between time complexity studies of metaheuristics and fitness landscape analysis. The concept of the escape probability is formally defined and a theoretical analysis is performed to investigate the relationship between the escape probability and the expected runtime, which is usually taken as a measure of problem difficulty in time complexity studies of metaheuristics [93]. Two further developments are obtained based on the escape probability. First, the fitness-probability cloud is defined to obtain an overall characterisation of fitness landscapes. Second, a predictive problem difficulty measure, the accumulated escape probability, is derived from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms.

The remainder of this chapter is organised as follows. Section 3.1 introduces the motivation of proposing the escape probability and investigates the relationship between the escape probability and the expected runtime. Section 3.2 defines the fitness-probability cloud for characterising fitness landscapes, and the accumulated escape probability (*aep*) to explicitly quantify the problem difficulty with respect to algorithms. Section 3.3 presents a detailed experimental study of the *aep* on measuring difficulties of four different problems with respect to the mutation-based $(\mu + \lambda)$ EAs.

3.2 The Relationship Between the Escape Probability and the Expected Runtime

Metaheuristics usually do not have any explicit access to the function for which an optimum is sought. The only way to obtain information on the unknown function is by evaluating different search points and an evaluation of the fitness function is the most costly part in optimisation. The runtime of a metaheuristic for optimising a fitness function is defined as:

Definition 3 *(Runtime) [54]*

Let a metaheuristic $\mathcal{A}$ and a fitness function $f : X \to \mathbb{R}$ be given, X is a set of candidate solutions. The runtime $T_{\mathcal{A}}(f)$ of $\mathcal{A}$ on f is defined as the smallest number of function evaluations t such that $f(x_t) = max\{f(x)\}$.

The runtime of a metaheuristic for optimising a fitness function in general is a random variable. Its expectation $E(T_{\mathcal{A}}(f))$, the expected runtime of $\mathcal{A}$ on f, is usually taken as a measure of difficulty of f for $\mathcal{A}$ [54]. In time complexity studies of metaheuristics, rigorous runtime analysis for studying the expected runtime of metaheuristics has emerged as a solid theoretical means to understand how metaheuristics work and get to know their capabilities and limitations. The results can be used to judge the difficulty of problems with respect to different algorithms in a rigorous way. Runtime analyses have been performed for many pseudo-Boolean functions [125] as well as for many problems from combinatorial optimisation [92].

Markov chains are widely used mathematical models in runtime analysis of metaheuristics. The sequence of random variables $\{\xi_t; t = 0, 1, 2, \cdots\}$ can be modelled by a Markov chain, since the offspring solution often depends only on the parent solution [106]. For any states x, y in the search space, the transition probability $P(x, y; t)$ is given by:

$$P(x, y; t) := P(\xi_{t+1} = y)|\xi_t = x) \tag{3.1}$$

A metaheuristic with elitism strategy (the best fitness value in the population never decreases) can be modelled as an absorbing Markov chain. Let $\{\xi_t; t = 0, 1, 2, \cdots\}$ be an associated Markov chain with a meta-heuristic, the transition probabilities from the initial solution/state to the optimal solution/state can be used to estimate the expected runtime (the number of function evaluations to reach the optimum).

The fitness-level method, also known as the method of f-based partitions [125], is a direct approach used for proving upper bounds of expected runtime [111]. Assuming maximisation, the fitness-level method for proving upper bounds is introduced below.

Definition 4 *(Fitness-level method for proving upper bounds) [112]*

For two sets $A, B \subseteq \{0,1\}^n$ and fitness function f let $A < B$ if $f(a) < f(b)$ for all $a \in A$ and all $b \in B$. Consider a partition of the search space into non-empty sets $A_1, \cdots, A_m$ such that $A_1 < A_2 < \cdots < A_m$ and A_m only contains global optima. For a metaheuristic $\mathcal{A}$ we say that $\mathcal{A}$ is in A_i or on level i if the best solution created so far is in A_i. Consider $\mathcal{A}$ with elitism strategy and let s_i be a lower bound on the probability of creating a new offspring in $A_{i+1} \cup \cdots \cup A_m$, provided $\mathcal{A}$ is in A_i. Then the expected optimization time of $\mathcal{A}$ on f (without the cost of initialization) is bounded by

$$\sum_{i=1}^{m-1} P(\mathcal{A} \quad starts \quad in \quad A_i) \sum_{j=i}^{m-1} \frac{1}{s_i} \leq \sum_{i=1}^{m-1} \frac{1}{s_i} \tag{3.2}$$

The canonical partition is the one in which A_i contains exactly all search points with fitness i.

From the fitness-level method, it is noted that the probability of creating a new offspring on higher levels $\{i+1, \ldots, m\}$, provided $\mathcal{A}$ is on level i, is critical in determining general upper bounds of the expected runtime.

Inspired by this, the probability of creating a new offspring on higher levels $\{i+1, \ldots, m\}$ provided $\mathcal{A}$ is on level i is formally defined as the escape probability. And the relationship between the escape probability and the expected runtime is further investigated.

3.2.1 Preliminaries

The search process of a metaheuristic with elitism strategy has been modelled as a time-homogeneous Markov chain, where the transition matrix P is time-independent and the $k-step$ transition matrix P^k can be computed as the $k-th$ power of the transition matrix P. The notations used in the analysis are defined below.

- Runtime m_i: Average time of a metaheuristic to reach the optimum of a fitness function f when search starts from state i, $i = 1, 2, \dots, n$. Assuming n states in the search space, each state contains a set of similar solutions. The canonical partition is applied in which state i contains exactly all solutions with fitness f_i.

- Escape probability $p_i^e = \sum\limits_{j:f_j>f_i} p_{ij}$, $i, j = 1, 2, \dots, n$: p_i^e represents the probability of escaping from state i to another state of better fitness.

- Column sum in probability transition matrix $\mathbf{P}$: $c_j = \sum_{i=1}^{n} p_{ij}$, where $i = 1, 2, \dots, n$;

- Let $\mathbf{P}^{(k)} = \mathbf{P}^k$, and $c_j^{(k)} = \sum\limits_{i=1}^{n} p_{ij}^{(k)}$, where $i, j = 1, 2, \dots, n$, and $k = 1, 2, 3, \dots$

3.2.2 Investigating the Relationship Between the Escape Probability and the Expected Runtime

This section presents the theoretical investigation on the relationship between the escape probability p_i^e and the expected runtime $\exp(m_i)$ [81].

Equation 3.3 below shows that m_i can be expanded as state i and j are adjacent:

$$m_i = \sum_{j=1}^{n} m_j p_{ij} + 1, i = 1, 2, \ldots, n \tag{3.3}$$

Equation 3.3 can be rewritten as:

$$m_i = \sum_{j=1}^{n} (m_j + 1) p_{ij} \tag{3.4}$$

The sum of m_i:

$$\begin{aligned}
\sum_{i} m_i &= \sum_{i=1}^{n} \sum_{j=1}^{n} (m_j + 1) p_{ij} \\
&= \sum_{j=1}^{n} \sum_{i=1}^{n} (m_j + 1) p_{ij} \\
&= \sum_{j=1}^{n} m_j \sum_{i=1}^{n} p_{ij} + \sum_{j=1}^{n} \sum_{i=1}^{n} p_{ij} \\
&= \sum_{j=1}^{n} m_j c_j + n
\end{aligned} \tag{3.5}$$

In Equation 3.6 below, m_i is computed as an expectation of the runtime for the search to escape from state i to another state j at different times. $t \times (1 - p_i^e)^{t-1} p_i^e + \sum_{j \neq i} m_j p_{ij}^{(t)}$ represents the runtime for the search to escape from state i to state j

at time t. m_i can be expanded as:

$$\begin{aligned} m_i = 1 \times p_i^e + \sum_{j \neq i} m_j p_{ij}^{(1)} \\ + 2 \times (1 - p_i^e)^1 p_i^e + \sum_{j \neq i} m_j p_{ij}^{(2)} \\ + \dots \\ + t \times (1 - p_i^e)^{t-1} p_i^e + \sum_{j \neq i} m_j p_{ij}^{(t)} \\ + \dots \end{aligned} \tag{3.6}$$

Summing both sides of the Equation 3.6, the following is obtained:

$$\begin{aligned} \sum_{i=1}^{n} m_i &= \sum_{i=1}^{n} 1/p_i^e + \sum_{t>0} (\sum_{j=1}^{n} m_j (c_j^{(t)} - p_{jj}^{(t)})) \\ &= \sum_{i=1}^{n} 1/p_i^e + \sum_{j=1}^{n} m_j \sum_{t>0} (c_j^{(t)} - p_{jj}^{(t)}) \end{aligned} \tag{3.7}$$

Given that $c_j^{(t)} - p_{jj}^{(t)} \geq 0$, the following can be obtained:

$$\begin{aligned} \sum_{i=1}^{n} m_i &\geq \sum_{i=1}^{n} 1/p_i^e + \sum_{j=1}^{n} m_j (c_j - p_{jj}) \\ &\geq \sum_{i=1}^{n} 1/p_i^e + \sum_{j=1}^{n} m_j c_j - \sum_{j=1}^{n} m_j p_{jj} \end{aligned} \tag{3.8}$$

By solving the system of equations formed by equations 3.5 and 3.8:

$$\begin{cases} \sum_{i=1}^{n} m_i \geq \sum_{i=1}^{n} 1/p_i^e + \sum_{j=1}^{n} m_j c_j - \sum_{j=1}^{n} m_j p_{jj} \\ \sum_{i=1}^{n} m_i = \sum_{j=1}^{n} m_j c_j + n \end{cases} \tag{3.9}$$

The following can be obtained:

$$\sum_{i=1}^{n} m_i p_{ii} \geq \sum_{i=1}^{n} 1/p_i^e - n \tag{3.10}$$

$$\sum_{i=1}^{n} m_i \geq \sum_{i=1}^{n} m_i p_{ii} \geq \sum_{i=1}^{n} 1/p_i^e - n \tag{3.11}$$

The relationship between the escape probability and the expected runtime is given below:

$$\exp(m_i) \geq \frac{\sum_{i=1}^{n} 1/p_i^e}{n} - 1 \tag{3.12}$$

Equation 3.12 gives a general lower bound of the expected runtime $\exp(m_i)$, which is determined by the escape probability p_i^e and a constant n.

3.3 Fitness-Probability Cloud for Characterising Fitness Landscapes and Measuring Problem Difficulty

Given that the escape probability can determine general lower bounds of the expected runtime, the escape probability shows great promises to serve as a reliable indicator of the problem difficulty in relation to algorithms. To bridge the gap between the theoretical results on the escape probability and empirical fitness landscape analysis, two important developments are obtained. First, the fitness-probability cloud [82] is defined to obtain an overall characterisation of fitness landscapes. Second, a predictive problem difficulty measure, the accumulated escape probability [82], is derived from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms.

3.3.1 Escape Probability

In theoretical analysis, the escape probability which represents the probability for a metaheuristic $\mathcal{A}$ to reach higher fitness levels from a certain fitness level, can be computed by assuming the transition matrix is known. However, this assumption cannot be met in practical scenarios. The escape probability can be estimated in a way that the information about the transition matrix is not required, i.e., the escape probability at level i is taken as the average from the probability for $\mathcal{A}$ to reach better solutions from each solution at level i.

Definition 5 *(Escape Probability) [82]*

For two sets $A, B \subseteq \{0,1\}^n$ and fitness function f let $A < B$ if $f(a) < f(b)$ for all $a \in A$ and all $b \in B$. Consider a partition of the search space into non-empty sets $A_1, \cdots, A_m$ such that $A_1 < A_2 < \cdots < A_m$ and A_m only contains global optima. For a metaheuristic $\mathcal{A}$ we say that $\mathcal{A}$ is in A_i or on level i if the best solution created so far is in A_i.

The escape probability for $\mathcal{A}$ to escape from level i to levels $\{i+1, \cdots, m\}$ is defined as:

$$P_i^e = \sum_{s \in A_i} p_s^e, \tag{3.13}$$

where p_s^e represents the escape probability of a solution $s \in A_i$ to reach better solutions by applying one step of $\mathcal{A}$. Obviously, the higher the value of P_i^e, the easier to escape from level i for $\mathcal{A}$.

3.3.2 Fitness-Probability Cloud

The escape probability lays the foundation to develop a novel approach for characterising fitness landscapes and measuring problem difficulty based on it. Based on the notion of escape probability, a natural way to study whether a problem is difficult or easy for a given algorithm, is to plot the fitness values against the escape probabilities, where the escape probability is obtained for each fitness level. This plot is formally defined as the fitness-probability cloud.

Definition 6 *(Fitness-Probability Cloud) [82]*

Given the definition of P_i^e as the escape probability from fitness level i to a better fitness level, for fitness function f with m distinct fitness levels, the fitness-probability cloud is defined as a set of points on a bi-dimensional plane:

$$fpc = \{(f_1, P_1^e), (f_2, P_2^e) \ldots, (f_m, P_m^e))\} \tag{3.14}$$

Fitness-probability cloud gives an overall characterisation of the underlying fitness landscape, and thus indicating the problem difficulty with respect to the algorithm applied to optimise it. In particular, the shape of the fitness-probability cloud can give some hints on understanding behaviours of different algorithms. For example, assuming maximisation, if the escape probability on a fitness-probability cloud starts out very high but drastically decreases to a very low level as the fitness value increases, we can hypothesize that the algorithm can efficiently find approximation solutions, but is not very effective in locating the optimal solution of the problem; in contrast, if the escape probability on a fitness-probability cloud starts out at a relatively low level but increases as the fitness value increases, we can hypothesize that the algorithm seems inefficient to find approximation solutions, but shows promise in finding the optimal solution of the problem.

3.3.3 Measuring the Problem Difficulty: Accumulated Escape Probability

The fitness-probability cloud provides an illustrative characterisation of the underlying fitness landscape, which gives a straightforward indication of the problem difficulty with respect to the algorithm applied to optimise it. Also the fluctuations of the escape probability over fitness can give some hints on better understanding strengths and weaknesses of search algorithms. However, the mere observation of its shape still lacks formality, also it is very difficult for practitioners to select the best suited algorithm by means of comparing the fitness-probability clouds generated from different algorithms.

Therefore, an algebraic measure needs to be derived which can capture the critical information on the fitness-probability cloud. The definition of such a measure also opens up the possibility of incorporating it to build enhanced techniques for finding the best suited algorithm for solving a particular problem. In fact many forms of algebraic measure can be derived from the fitness-probability cloud. Here a first attempt is made to define the accumulated escape probability to explicitly measure the problem difficulty with respect to algorithms. The accumulated escape probability is formally defined as:

Definition 7 *(Accumulated Escape Probability) [82]*

$$aep = \sum_{i=1}^{m} w_i P_i^e, \tag{3.15}$$

where (w_i) is a set of weights which determine the contributions of the escape probabilities from different fitness levels to the final aep measure. The aep measures problem difficulty with respect to algorithms in the following way: the lower the value of aep, the more difficult the problem is with respect to the algorithm.

The definition of *aep* is general and has many degrees of freedom as different forms of (w_i) can be considered. In its simplest form, if the escape probabilities from different fitness levels are taken as equally important, the *aep* measure is defined as the average escape probability for all fitness levels:

$$aep = \frac{\sum_{i=1}^{m} P_i^e}{m} \tag{3.16}$$

3.3.4 Sampling Method

In this section, the sampling method to estimate escape probability and generate the fitness-probability cloud is described. For an NP-hard optimisation problem, the size of the search space is exponentially large which does not allow consideration of all candidate solutions, the only feasible approach is to use samples. To estimate the escape probability P_i^e, a set of solutions with different fitness values is needed. Since sampling is computationally expensive, a compact yet representative set of samples is desired. It is noted that not all solutions in the search space are equally important, for example, good-quality solutions play a more crucial role in determining the overall difficulty of the problem, therefore it is preferred to sample the search space according to a distribution that gives higher weights to good-quality solutions. This can be achieved by any importance sampling method such as the Metropolis method [83]. In this case, the Metropolis-Hastings sampling method is selected which is an extension of Metropolis to non-symmetric stationary probability distributions [117].

Let f be the fitness function, the procedure to generate a set of samples $\{s_1, s_2, \cdots, s_n\}$ using the Metropolis-Hastings sampling method is described in Algorithm 1.

To estimate the escape probability, for each sampled solution, a set of neighbouring solutions are generated by applying one step of the search operator/heuristic. The escape probability of a sampled solution is estimated as the proportion of neighbouring solutions with better fitness values out of the sampled neighbourhood set. And the escape probability P_i^e is then estimated as the mean value of the escape probabilities of sampled solutions with fitness value f_i. Hereafter $\{f_1, \cdots, f_m\}$ is

Algorithm 1: Metropolis-Hastings sampling method

```
begin
    $\alpha(x, y) = \min(1, \frac{y}{x})$
    $s_1$ is sampled uniformly at random;
    $i = 1$;
    while $i < n$ do
        1. a solution $\theta$ is sampled uniformly at random;
        2. a random number $u$ is generated from a uniform (0,1) distribution;
        3. if $u \leq \alpha(f(s_i), f(\theta))$ then
            $s_{i+1} \leftarrow \theta$;
        end
        else
            Go to 1;
        end
        $i++$;
    end
end
```

referred to as a set of distinct fitness values in the sampled solutions obtained using the Metropolis-Hastings method, and for each f_i, P_i^e is the estimated escape probability computed from the sampled neighbourhood set.

3.4 Experimental Studies on Unination Functions and Subset Sum Problem w.r.t. $(\mu + \lambda)$ EAs

In this section, a detailed experimental study is presented to evaluate the performance of the proposed predictive problem difficulty *aep*. The effectiveness of *aep* as a predictive difficulty measure is verified with experiments on predicting the performance of mutation-based $(\mu + \lambda)$EAs on the unitation functions and the subset sum problem. First, the unination functions [88] and the subset sum problem are introduced, where unitation functions are a set of benchmark functions extensively studied in the literature, and the subset sum problem is a well studied NP-hard problem. Second, the performance of *aep* is demonstrated by verifying the performance predictions given by *aep* with the actual performance measured by a standard a posteriori performance measure. Furthermore, the performance predictions given by a widely used predictive problem difficulty measure, the negative slope coefficient (nsc), are obtained for comparison.

3.4.1 Test Problems

3.4.1.1 Unitation Functions

Three unitation functions: OneMax, Trap, OneMix [88] are used.

Definition 8 *Let s be a bit string of length l, the unitation $u(s)$ of s is a function defined as:* $u(s) = \sum_{i=1}^{l} s_i$.

OneMax functions are generalisations of the unitation $u(s)$ of a bit string s:

$f(s) = d \cdot u(s)$, where d is 1.

Trap function[30] is defined as follows:

$$f(s) = \begin{cases} \frac{a}{z}(z - u(s)), & if \quad u(s) \leq z \\ \frac{b}{l-z}(u(s) - z), & otherwise \end{cases} \tag{3.17}$$

where a represents a local optimum and b is a global optimum, z is a slope-change location.

OneMix function is a mixture of OneMax function and ZeroMax function, which is formally defined as:

$$f(s) = \begin{cases} (1+a)(\frac{l}{2} - u(s)) + \frac{l}{2}, & if \quad g(s) \\ u(s), & otherwise \end{cases} \tag{3.18}$$

where a represents a constant above zero and $g(s)$ is equal to 1 when $u(s)$ is even and $u(s) < \frac{l}{2}$.

3.4.1.2 Subset Sum Problem

The Subset Sum problem is a constrained optimisation problem. Given a set of n items each with an associated weight w, the problem is to select a subset out of n items, where the weighted sum is maximised and does not exceed the budget W. Mathematically this problem is formulated as follows:

$$\text{Maximise} \sum_{i=1}^{n} w_i x_i,$$

$$\text{Subject to} \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}, \quad W = \frac{\sum_{i=1}^{n} w_i}{2}$$

3.4.2 Experimental Setup

For each of the four test problems: OneMax, Trap, OneMix and the Subset Sum problem, four problem instances with varying problem size from 20 to 200 are used throughout the experiments.

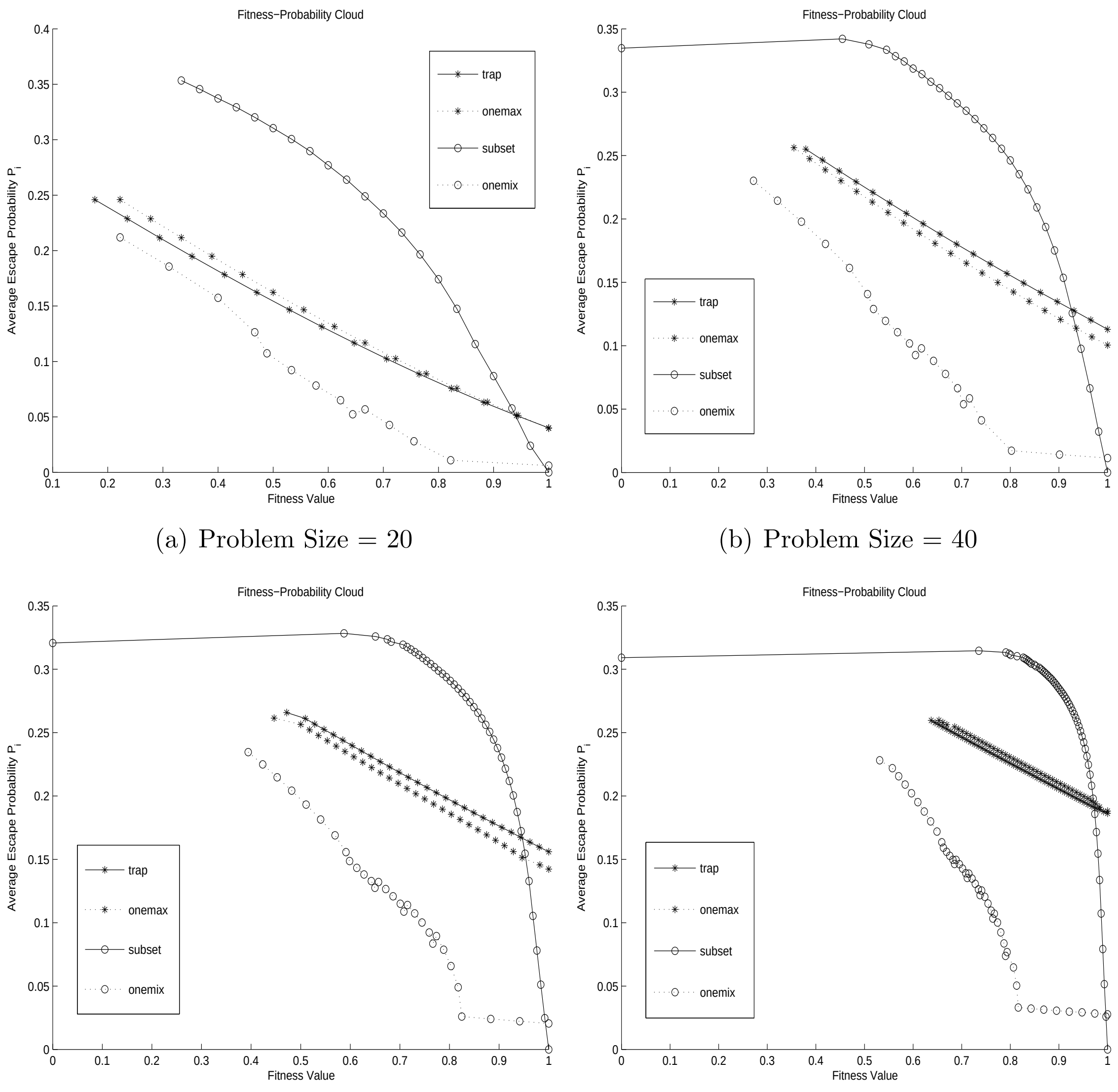
The target algorithm is the mutation-based $(\mu+\lambda)$ EAs (μ denotes the number of parent solutions in the population, and λ denotes the number of offspring solutions in the population). A population of μ parents generate λ offspring, then the best μ solutions in μ parents and λ offspring are selected as the next generation. The $(\mu + \lambda)$ EA uses only a mutation operator, no crossover is involved. The mutation operator is the bitwise mutation with flip probability $1/n$, n is the problem size. Three different configurations of the target algorithm are considered.

Evaluating a predictive problem difficulty measure requires suitable metrics for evaluating the performance of metaheuristics on problem instances. The performance metrics are used to validate whether the performance predictions given by a predictive problem difficulty measure are correct or not. For any two problems, the metrics should distinguish the relative difficulty of solving the problems by a given algorithm. And equally, given two algorithms, the metrics should distinguish the relative difficulty in the algorithms solving the same problem. A common way of measuring the performance of metaheuristics is using the number of function evaluations required by an algorithm to find the solution for a problem instance. For real-world problems where the optimal solutions are unknown, the number of function evaluations consumed by the algorithm upon satisfying the stopping criteria is taken as the performance metric. To ensure fair comparisons between different algorithms, a sufficient number of independent runs of each algorithm needs to be performed and usually the average number of function evaluations is taken as the metric for comparison.

In our experiments, the performance of the mutation-based $(\mu + \lambda)$EAs are measured by the number of function evaluations taken upon reaching the stopping criteria, i.e., there is no improvement in terms of the best solution found in 500 function evaluations. This is determined by preliminary experiments which showed that the algorithm converged with no progress in 500 function evaluations. To obtain the performance metrics for the mutation-based $(\mu + \lambda)$ EAs on four test problems with problem size 20, 40, 80 and 200. For each problem instance, 100 independent executions are performed and the mean number of function evaluations is taken as the performance metric for comparison.

3.4.3 Experimental Results

First of all, the fitness-probability clouds for the test algorithm and test problem instances have been generated. To be more specific, for each fitness-probability cloud, 1000 samples were obtained from the search space using the Metropolis-Hastings sampling method. For each sampled solution, by applying one step of the bitwise mutation operator with flip probability $1/n$, 10000 solutions in the neighbourhood are sampled for estimating its escape probability. The fitness-probability clouds generated for four test problems of problem sizes 20, 40, 80 and 200 w.r.t. ($\mu + \lambda$) EAs with the bitwise mutation operator (flip probability $1/n$) are illustrated in Figure 3.1.



(a) Problem Size = 20 (b) Problem Size = 40

(c) Problem Size = 80 (d) Problem Size = 200

Figure 3.1: Plot of Fitness-Probability Clouds for Four Test Problems with Problem Size 20, 40, 80 and 200 [82].

Based on the generated fitness-probability clouds, the corresponding values of the accumulated escape probability (aep) can be derived. For the sake of comparison,

Table 3.1: Actual Performance Measured by the Number of Function Evaluations vs. *aep* Predictions vs. *nsc* Predictions of Three Different (μ+λ) EAs on Four Test Problems with Size 20, 40, 80 and 200 [82].

Problem	Problem Size	(1+1) EA	(3+7) EA	(7+3) EA	*aep*	*nsc*
OneMax	20	641	1166	1110	0.135	0
Trap	20	627	1158	1105	0.135	0
OneMix	20	745	1375	1330	0.09	-8.1932
Subset Sum	20	548	1009	928	0.22	-1.1572
OneMax	40	821	1434	1430	0.175	-0.333
Trap	40	829	1422	1438	0.182	0
OneMix	40	1028	1776	1728	0.105	-16.3114
Subset Sum	40	533	1009	928	0.239	-6.818
OneMax	80	1267	2002	2134	0.202	-0.5
Trap	80	1273	2004	2115	0.209	-0.25
OneMix	80	1609	2608	2678	0.121	-20.4879
Subset Sum	80	547	1015	936	0.246	-7.5286
OneMax	200	2640	3848	4221	0.225	-3
Trap	200	2590	3860	4242	0.222	0
OneMix	200	3070	4724	4952	0.121	-30.175
Subset Sum	200	534	1021	945	0.252	-8.6169

aep has been compared with the negative slope coefficient (*nsc*) [119], which has been widely tested on a variety of Genetic Programming (GP) and Genetic Algorithm (GA) problems showing considerable reliability in distinguishing easy from hard problems. This is because *aep* and *nsc* are both predictive measures which do not require any domain-specific knowledge, whilst other previous measures such as the fitness-distance correlation and the epistasis variance require known global optima.

The *nsc* is defined based on the notion of fitness cloud [121], which is a set of points $C = \{(f_1, f_1^{'}), \cdots, (f_n, f_n^{'})\}$ on a bi-dimensional plane, where f is the fitness function, f_i denotes the fitness value of a solution and $f_1^{'}$ denotes the fitness value of its neighbours. C can be partitioned into a number of discrete bins $C_1, \cdots, C_m$

such that $(f_a, f_a') \in C_j$ and $(f_b, f_b') \in C_k$ with $j < k$ implies $f_a < f_b$. For each of the line segments defined between the centroids of adjacent bins, a slope S_i is defined as:

$$S_i = \frac{f_{i+1}' - f_i'}{f_{i+1} - f_i} \tag{3.19}$$

Finally the negative slope coefficient is defined as:

$$nsc = \sum_{i=1}^{m-1} \min(0, S_i) \tag{3.20}$$

The hypothesis proposed in [119] is that nsc should classify the problem difficulty in the following way: $nsc = 0$ indicates an easy problem and smaller values indicate higher problem difficulty when $nsc < 0$.

The experimental results on the performance of the mutation based $(\mu + \lambda)$ EAs on all instances of the four test problems measured by the number of function evaluations, as well as the predictions of aep and nsc, are summarised in Table 3.1 [82].

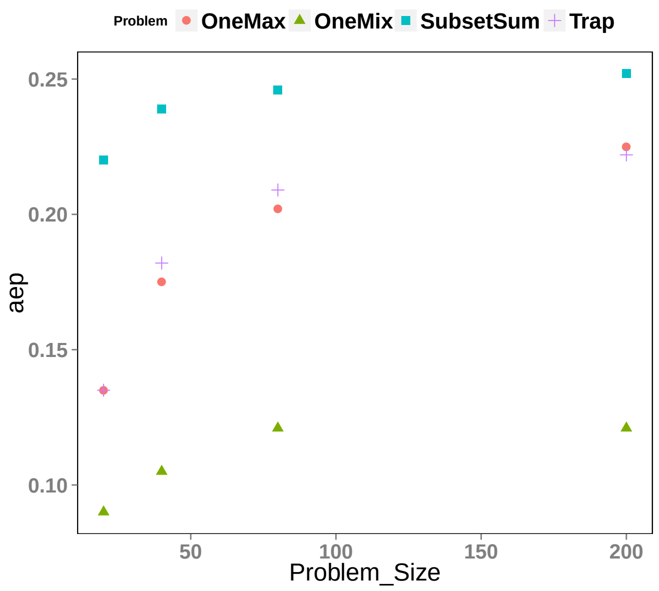
The third to fifth columns in Table 3.1 presented the performance of the mutation-based $(\mu+\lambda)$ EAs executed under three different configurations measured by the number of function evaluations. Each of the results is the mean value over 100 independent executions. According to the performance metric, i.e. the number of function evaluations, the relevant problem difficulty between four test problems (in the same problem size) w.r.t. the mutation-based $(\mu+\lambda)$ EAs under three different configurations are consistent. To be more specific, the order of problem difficulty given by the performance metric was: Subset Sum $<$ OneMax $\approx$ Trap $<$ OneMix.

The sixth column in 3.1 showed the results of the aep measure. By definition the smaller the value of aep, the more difficult the problem is for the algorithm.

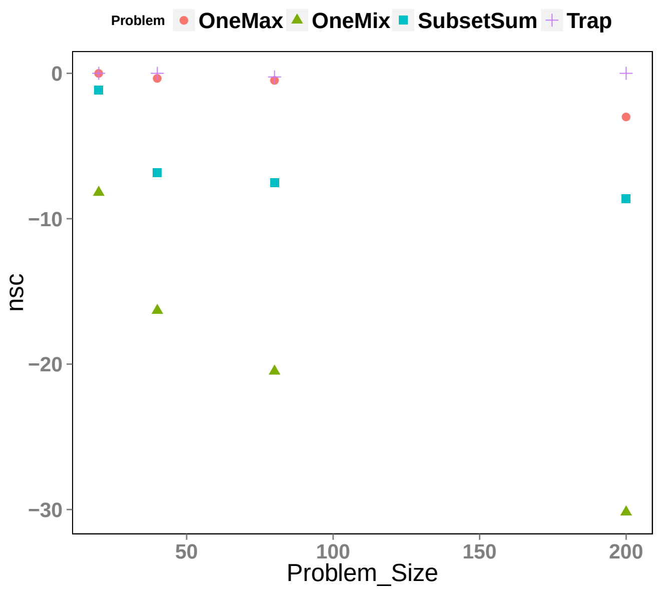
Applying this rule to judge the problem difficulty given by *aep*, one can observe that *aep* consistently gave the relative problem difficulty of the same size across four different problem sizes: Subset Sum $<$ OneMax $\approx$ Trap $<$ OneMix. The results were illustrated in Figure 3.2(a). It is clear to see that the results given by *aep* were in qualitative agreement with the results given by the performance metric.

In contrast to the correct predictions made by the *aep* measure, as can be seen from Table 3.1, the negative slope coefficient (*nsc*) gave an order of problem difficulty: OneMax $\approx$ Trap $<$ Subset Sum $<$ OneMix, as illustrated in Figure 3.2(b). This result failed to correspond to the order of problem difficulty given by the performance metric. As a consequence, the *nsc* measure seemed to be unable to correctly distinguish the relative problem difficulty between OneMax, Trap, OneMix and Subset Sum w.r.t. the mutation-based (μ+λ) EAs.

Since the *aep* measure was able to broadly discriminate easy and hard problems with reference to mutation-based (μ+λ) EAs, a further question is whether the *aep* measure was able to quantify the difference between various problems in terms of the problem difficulty w.r.t. the algorithm. In this case, if the problem instances of size 20 and (1+1) EA are taken as an example, the subset sum problem seemed to be the easiest, on one hand, in terms of the performance metric for (1+1)EA, the onemax and trap functions were approximately 17% harder than the subset sum, and the onemix function was about 36% harder. On the other hand, in terms of the *aep* values, the onemax and trap functions were approximately 38% harder than the subset sum and the onemix function was about 59% harder. Therefore, the problem difficulty between the onemix and the subset sum were twice as much as the that between onemax/trap and the subset sum, which was roughly the case as indicated by the *aep* measure. Consequently, the *aep* measure was able to quantify the relative difference in problem difficulty, however, it was unable to quantify the absolute difference in problem difficulty.



(a) Results of *aep* Measure on the Test Problems



(b) Results of *nsc* Measure on the Test Problems

Figure 3.2: Plot of aep and nsc measures for Four Test Problems of Problem Sizes 20, 40, 80 and 200.

3.5 Summary

In summary, this chapter has for the first time bridged the gap between time complexity studies of metaheuristics and fitness landscape analysis. In particular, the notion of escape probability originated from runtime analysis has been formally defined and a theoretical analysis to investigate the relationship between the escape probability and the expected runtime has been conducted. Based on the escape probability, two important developments have been obtained. First, the fitness-probability cloud has been defined to obtain an overall characterisation of fitness landscapes. Second, a predictive problem difficulty measure, the accumulated escape probability (*aep*), has been derived from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms.

aep is a predictive problem difficulty measure which does not require any domain-specific knowledge. It is hypothesized that *aep* should measure the problem difficulty with respect to a given algorithm in the following way: the smaller the *aep* value, the more difficult the problem is with respect to the algorithm. The effectiveness of the *aep* has been verified with experiments on measuring difficulties of four different problems including the subset sum problem and the unitation functions such as one-max, onemix and trap with respect to the mutation-based (μ+λ) EAs. The experimental results showed that *aep* can serve as a reliable predictive problem difficulty measure for discriminating the relative problem difficulty w.r.t. the mutation-based (μ+λ) EAs. For comparison purpose, the negative slope coefficient (*nsc*), a widely applied problem difficulty measure in GP and GA, has been applied to predict performance of the mutation-based (μ+λ) EAs on four test problems. This is due to that *aep* and *nsc* are both predictive measures which do not require any domain-specific knowledge, whilst other previous measures such as the fitness-distance correlation and the epistasis variance require known global optima. The results showed that *nsc* failed to correctly discriminate the relative problem difficulty of the subset sum problem, onemax, onemix and trap w.r.t. the mutation-based (μ+λ) EAs.

The proposed fitness-probability cloud and *aep* measure showed great promise in effectively characterising fitness landscapes and reliably measuring problem difficulty. Since the fitness-probability cloud and *aep* are generic approaches which do not require any domain-specific knowledge, also the definition of *aep* is general and has many degrees of freedom, the fitness-probability cloud and *aep* can serve as an effective and reliable approach for fitness landscape analysis, and can potentially be incorporated to build enhanced techniques for finding the best suited algorithm for solving a particular problem.

Chapter 4

INCORPORATING FITNESS LANDSCAPE ANALYSIS FOR STATIC ALGORITHM CONFIGURATION

4.1 Introduction

Metaheuristics are general algorithmic frameworks, even a metaheuristic algorithm can have many variants if different algorithm configurations are used. It is widely acknowledged that finding good algorithm configurations is essential to obtain robust and high algorithm performance. For example, a simulated annealing algorithm can perform extremely differently with different values of the temperature parameter even on the same problem instance. Furthermore, it has been observed that an algorithm requires different configurations in order to find good solutions for different problem instances [41, 94, 137].

As of yet, finding the best algorithm configuration for solving a particular problem instance remains one of the persisting challenges in the field of metaheuristics. This is due to the fact that there is in fact very limited understanding of which algorithms, or which algorithm variants are best suited for solving which kinds of problem instances. Previous work has revealed that a large fraction of time in algorithm development has been spent on configuring the algorithm [1, 11], since in many cases the algorithm is configured manually by trial and error, and the configuration space is essentially very large even for a handful of parameters [79]. Clearly it is not only inefficient and laborious, but very unlikely to locate the optimal algorithm configuration, if the algorithm were to be configured manually.

Recent years have seen many approaches proposed to automatically determine a priori the best suited configuration of a given algorithm for solving a particular problem instance. In general, most existing approaches proceed along two main modes:

- **Generic, one-size-fits-all Configuration Methods** Generic (problem-independent) methods which produce a one-size-fits-all configuration of a given algorithm for an entire set of problem instances.

- **Problem-dependent, per-instance Configuration Methods** Problem-dependent methods which perform algorithm configuration on a per-instance base. Each method only applies to one particular optimisation problem by making use of problem-specific features.

To the best of our knowledge, there exist very few generic (problem-independent) techniques for automatically configuring a given algorithm on a per-instance basis, that is, taking into account the characteristics of the problem instances to solve and past performance on similar instances. This chapter addresses this need via incorporating a problem difficulty measure, the accumulated escape probability, to build a generic approach which performs automatic algorithm configuration on a per-instance base. The proposed approach is based on learning the pattern which governs the relationship between the configurations of a given algorithm and the characteristics of problem instances, where the accumulated escape probability is employed as a feature to characterise problem instances. Further to determining the best suited configuration within a finite set of candidate configurations, it is always useful to produce a novel, problem-specific configuration (e.g. a new search operator/heuristic) that potentially outperforms other configurations on a particular class of problem instances. This chapter establishes a bridge connecting results of theoretical fitness landscape analysis and the design of novel, effective heuristics, through proposing an approach to perform elementary landscape analysis and further to explicitly apply the analytical results to build novel and effective local search heuristics. This chapter proceeds as follows. Section 4.1 first reformulates the static algorithm configuration problem as a decision problem to be studied, followed by the descriptions of the proposed automatic, per-instance algorithm configuration method developed with learning the pattern between the configurations of a given algorithm and the problem instances represented by the accumulated escape probability. The utility of the proposed approach is illustrated by automatically configuring the $(\mu + \lambda)$ EAs for solving the unique input output sequence problem (UIO) in software testing [74]. Section 4.2 presents the theoretical results arising from studies of elementary landscape, and proposes the approach to perform elementary landscape analysis and further to explicitly apply the analytical results for developing enhanced search heuristics. The proposed approach is applied to develop a novel and effective stochastic hill climbing algorithm for solving the next release problem (NRP) [8]. Section 4.3 summarises this chapter with a discussion.

4.2 Incorporating Problem Difficulty Measure to Build Automatic Algorithm Configuration Methods

4.2.1 Motivation

Given an algorithm to solve a combinatorial optimisation problem, finding good configurations of the algorithm is essential to obtain robust and high algorithm performance. Manual methods are not only inefficient and laborious, but very unlikely to find the optimal algorithm configuration. This motivates the development of automatic algorithm configuration methods. Several generic approaches have been proposed which returns a one-size-fits-all configuration for an entire set of instances. However, different instances may require the algorithm to use very different configurations in order to find good solutions. In contrast, the approaches which perform automatic algorithm configuration on a per-instance basis only apply to particular problems by making use of problem-specific features.

An interesting research question is whether there are patterns or rules governing the choice of algorithm configurations, and whether such patterns can be learnt. This chapter proposes a generic (problem-independent) approach to perform automatic algorithm configuration on a per-instance base. The proposed approach employs classification algorithms to learn the pattern which governs the relationship between the configurations of a given algorithm and the characteristics of problem instances, where the accumulated escape probability is used to characterise the problem instances.

4.2.2 Reformulating Static Algorithm Configuration As a Decision Problem

The static algorithm configuration problem is commonly formulated as an optimisation problem, where the overall goal of the problem is to find a configuration θ such that the cost function $c(\theta)$ is optimised. However, unlike standard optimisation problems, this problem cannot be optimised directly since the cost function $c(\theta)$ cannot be written analytically and is typically highly non-linear and very expensive to compute. Therefore, it is unlikely to explore the entire configuration space for finding the optimal configuration within reasonable time.

Due to the fact that the static algorithm configuration problem is intractable to be solved to optimality, a more practical scenario to consider is how to find approximation solutions to this problem efficiently. This subsection reformulates the static algorithm configuration problem as a decision problem, which determines whether a given algorithm configuration θ is suitable for solving a particular problem instance.

To avoid potential confusions between the algorithm to be configured and the algorithm developed to undertake the configuration task, the former is referred to as the target algorithm and the latter as algorithm configuration method [66]. Let $\mathcal{A}$ denote the target algorithm with a set of parameters P which can be numerical, ordinal (large, medium, small) or categorical (choice of search operator/heuristic). The configuration space Θ consists of the domain of values for each parameter $p \in P$. Let the vector $\theta = \{p_1 \cdots p_k\}$ denote a configuration of $\mathcal{A}$. $\mathcal{H}$ is a function to measure the performance of the target algorithm executed under a given configuration θ for solving a problem instance. The evaluation function $\mathcal{H}$ can be in many different forms, for example, we might be interested in minimising the runtime of the target algorithm for finding a solution or maximising the quality of the solution found within fixed time. The overall goal of an algorithm configuration method is to find a configuration θ for each problem instance $I \in \mathcal{D}$, where the performance of the target algorithm $\mathcal{A}$ over $\mathcal{D}$ is optimised.

The static algorithm configuration problem considered in this thesis is formulated as a decision problem:

Definition 9 *(Static Algorithm Configuration As a Decision Problem).*

An instance of the algorithm configuration problem is a tuple $\langle \mathcal{A}, P, \Theta, \mathcal{D}, \mathcal{H}, m \rangle$, where:

- *$\mathcal{A}$: A parameterised algorithm;*
- *P: Parameters of $\mathcal{A}$;*
- *Θ: Configuration space of P;*
- *I: A problem instance;*
- *$\mathcal{H}$: A function that measures the performance of running $\mathcal{A}(\theta)$, $\theta \in \Theta$ with random seed rs on I;*
- *RS: A set of random seeds;*
- *$\mathcal{T}$: A threshold which determines if the performance of a configuration θ in terms of $\mathcal{H}$ is satisfactory.*
- *m is a statistical parameter. (Examples are expectation, mean, and variance.)*

The cost function for any candidate solution θ is:

$$c(\theta) = m_{rs \in RS}[\mathcal{H}(\mathcal{A}, \theta, I, rs)] \tag{4.1}$$

Assuming minimising $\mathcal{H}$, for a configuration θ and a problem instance I, the static algorithm configuration problem is to determine whether the performance of θ given by $c(\theta) = m_{rs \in RS}[\mathcal{H}(\mathcal{A}, \theta, I, rs)]$ is satisfactory or not, i.e., whether $c(\theta) < \mathcal{T}$.

4.2.3 Problem Difficulty Measures

This section introduces the problem difficulty measure, the accumulated escape probability, to be incorporated in building a generic approach that can perform automatic algorithm configuration on a per-instance base, and summarises the definitions of fitness-probability cloud, accumulated escape probability and the sampling method to estimate them in practice.

The accumulated escape probability (aep) [82] is defined based on the fitness-probability cloud [82]. The fitness-probability cloud (fpc) is a metaphor to obtain an overall characterisation of fitness landscapes which can give an indication of the problem difficulty with respect to the algorithm applied to optimise it. For a thorough characterisation of a problem instance, multiple fitness-probability clouds are required. This is because a problem instance can have a number of different fitness landscapes under different neighbourhood operators, and a fitness-probability cloud can only provide a partial characterisation of a problem instance, as the instance can exhibit different characteristics on the fitness-probability cloud when different neighbourhood operators are applied.

For the sake of extracting features from a problem instance, a set of fitness-probability clouds are required to be generated using different neighbourhood operators, where a set of aep measures computed based on the fitness-probability clouds generated are taken as the features to characterise the problem instance. It is hypothesized that there is a tight correlation between fitness-probability cloud and characteristics of problem instances, and a tight correlation between characteristics of problem instances and algorithm performance, therefore similar problem instances under the representation of aep measures should require similar configurations of a given algorithm to obtain robust and high algorithm performance.

4.2.3.1 Fitness-Probability Cloud and Accumulated Escape Probability

Escape Probability P_i^e, as defined in Definition 5, represents the probability for a metaheuristic $\mathcal{A}$ to reach higher fitness levels from a certain fitness level. Based on the notion of escape probability, a natural way to study whether a problem is difficult or easy for a given algorithm, is to plot the fitness values against the escape probabilities, where the escape probability is obtained for each fitness level. Let f be a fitness function with m distinct fitness levels, the fitness-probability cloud (fpc), as in Definition 6, is a set of points on a bi-dimensional plane:

$$fpc = \{(f_1, P_1^e), (f_2, P_2^e) \ldots, (f_m, P_m^e))\} \tag{4.2}$$

Although a fitness-probability cloud can obtain an overall characterisation of fitness landscapes, the proposed automatic algorithm configuration method requires quantitative measures that can be used as features to characterise problem instances. A problem difficulty measure, the accumulated escape probability (aep), is proposed which contains critical information on the fitness-probability cloud. The original definition of aep in Definition 7 is general and has many degrees of freedom, in this case, a simple form of aep is used which regards escape probability at each fitness level equally.

$$aep = \frac{\sum_{i=1}^{m} P_i^e}{m} \tag{4.3}$$

For computing the fitness-probability cloud and the accumulated escape probability in practice, statistical approaches such as sampling have to be employed due to the huge size of search space. In this case, the Metropolis-Hastings sampling method is used which is described in Section 3.3.

4.2.4 Proposed Automatic Algorithm Configuration Method

In this section, the proposed approach for automatically configuring a given algorithm on a per-instance base is introduced as illustrated in Figure 4.1. The major components of the proposed approach: feature extraction, the classification algorithm and the procedures for training and testing are described.

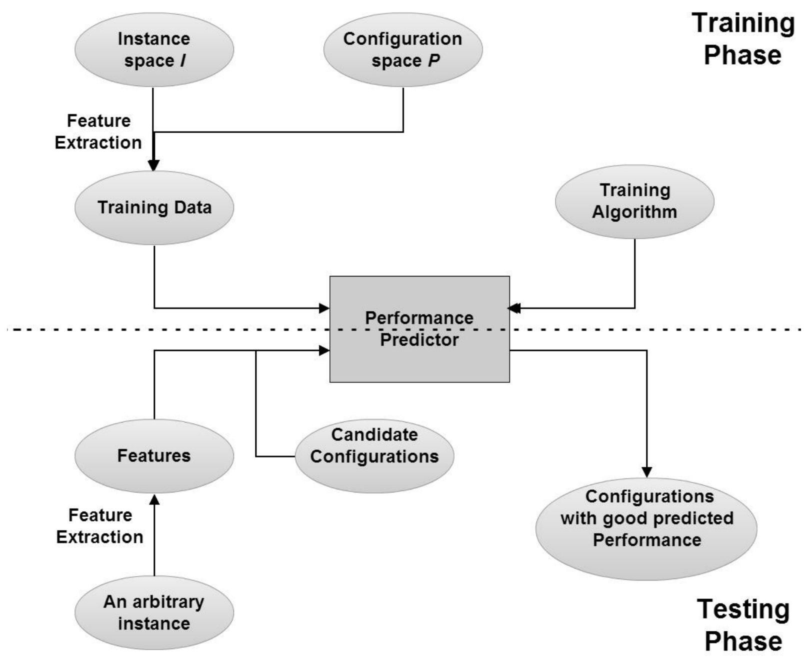


Figure 4.1: The Proposed Automatic Algorithm Configuration Method on a Per-instance Base.

4.2.4.1 Feature Extraction and Classification Algorithm

In the proposed automatic algorithm configuration method, the problem instances are represented as a set of *aep* measures computed based on a set of fitness-probability clouds generated using different neighbourhood operators. The overall goal is to train a classifier to learn the pattern which governs the choice of algorithm configurations, and thus identifying whether a configuration of the target algorithm is suitable for solving a problem instance. Since the learning is supervised, i.e., learning where a training set of correctly-identified observations is available. In this case, a training sample $\mathbf{D}$ consists of an instance's features and a configuration will be labelled as $L \in \{1, -1\}$, indicating whether the configuration is suitable for solving the instance or not.

A training sample consists of a problem instance's features and a configuration of the target algorithm, which is denoted as $\mathbf{D} = (\mathbf{PF}, \theta)$, where $\mathbf{PF}$ in $\mathbf{D}$ denotes the features of a problem instance and θ denotes a configuration of the target algorithm. In the proposed approach, problem instances are represented by a set of *aep* measures computed based on the fitness-probability clouds generated using different neighbourhood operators.

To determine the label for a training sample, for the target algorithm in a given configuration θ, a number of independent runs on the problem instance need to be performed where a statistical measure (e.g. mean number of the function evaluations) is taken to determine the algorithm performance. According to a predefined threshold on the algorithm performance, the label L is then determined as $L \in \{1, -1\}$, where 1 represents the configuration of the target algorithm is suitable for the problem instance, and -1 not.

The choice of the classification algorithm is essential to the performance of the proposed automatic algorithm configuration method. Typically the classification algorithm is selected based on the characteristics of the training data. In this case, the size of training samples is relatively small especially at the beginning since repeated executions of the target algorithm in different configurations are needed for each problem instance to generate training samples which is quite time-consuming. Also the training data is expected to be imbalanced since the training samples with the label $L = 1$ can be much less than those with the label $L = -1$.

In light of these characteristics of the training data, support vector machine

(SVM) [25] is selected as the classification algorithm. SVM has strong theoretical foundations and excellent empirical successes, which is developed on the basis of statistical learning theory and structural risk minimization and can handle small samples, nonlinear, high dimensionality, over learning and local minimum. Support vector machines have been considerably developed in pattern recognition, regression analysis, function estimator and time series prediction [33].

In the binary classification setting, SVM is a non-probabilistic binary linear classifier which takes a set of input data and predicts, for each given input, which of two possible classes forms the output. Given a set of training data $\{x_1, \cdots, x_n\}$ that are vectors in some space, and their labels $\{y_1, \cdots, y_n\}$ where $y_i \in \{-1, 1\}$, an SVM classification algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the training data are separated by a maximal margin. New examples are then mapped into that same space and predicted to belong to a category according to which side they fall on.

4.2.4.2 Training and Testing Phases

The overall steps in training and testing phases are summarised in Algorithm 2. In the training phase, the features of a problem instance, i.e., a set of *aep* measures are first obtained. The label attached to a training sample is determined by executing the target algorithm under a given configuration on a given problem instance. This procedure is repeated until the entire training data set is built. Then a classifier is obtained by running the classification algorithm on the training data, which can be used to determine whether a configuration of the target algorithm is suitable for solving a given problem instance. In the testing phase, the classifier takes input a configuration of the target algorithm and features of a given problem instance, i.e., a set of *aep* measures, and predicts whether the configuration is suitable for solving

Algorithm 2: Procedures for Training and Testing

```
Training Phase
Inputs:
$\mathcal{A}$: the target algorithm;
$I$ : a set of training instances;
$\Theta_t$: a set of configurations;
Outputs:
$C$: A classifier to determine if a given algorithm configuration is suitable for
solving a problem instance;
Procedure:
begin
    Let $PF$ = Features of an instance in $I$;
    Let $L = \{-1, 1\}$
    Let $F$ = a function to evaluate performance of the target algorithm
    executed under a configuration on an instance;
    for $inst \in I$ do
        for $\theta \in \Theta_t$ do
            $F$ = Execute target algorithm $\mathcal{A}$ under configuration $\theta$ on instance
            $inst$;
            $L$ = Attach the label to $(PF, \theta)$ according to $F$;
            $training\_sample(i) = (PF, \theta, L)$
            $i++$;
        end
    end
    $C$ = Run classification algorithm on the $training\_sample$ ;
    Output $C$;
end

Testing Phase
Inputs:
$C$: the classifier;
$inst$: an arbitrary instance;
$\theta$: a configuration in $\Theta$;
Outputs:
$L$: Performance prediction of the configuration $\theta$ on $inst$;
Procedure:
begin
    Let $PF$ = Features of $inst$;
    $L$ = Feed ($PF$, $\theta$) to the classifier $C$;
    Output $L$;
end
```

the problem instance or not.

4.2.5 Automatically Configuring $(\mu + \lambda)$ Evolutionary Algorithms for Solving the Unique Input Output Sequence Problem

The utility of the proposed automatic algorithm configuration method is illustrated by configuring the $(\mu + \lambda)$ evolutionary algorithms (EAs) for solving the unique input output sequence problem. First, the unique input output sequence problem is introduced. Second, the target algorithm $(\mu + \lambda)$ EA is described with the domains of possible values for its parameters. Third, the performance metrics to evaluate the performance of the target algorithm and the proposed automatic algorithm configuration method are defined, followed by the descriptions of the training data set.

4.2.5.1 Unique Input Output Sequence Problem

Finite state machines (FSMs) have been widely used to model software, communication protocols and circuits [74]. In FSM–based testing, a standard test strategy consists of two parts, namely, transition test and tail state verification. The former part aims to determine whether a transition of an implementation under test (IUT) produces the expected output while the latter checks that the IUT arrives at the specified state when a transition test is finished. Nearly all FSMs have Unique Input/Output Sequences (UIOs) for each state [2], and UIOs are the most widely used technique to generate robust and compact test sequences in finite state machine (FSM) based testing.

Computing UIOs is an NP-hard problem [74]. Lee and Yannakakis [74] note that adaptive distinguishing sequences and UIOs may be produced by constructing a state splitting tree. However, no rule is explicitly defined to guide the construction of an input sequence. Metaheuristics have proven efficient and effective in providing good solutions to some NP-hard problems in software engineering. Similarly to other problems in search-based software engineering [53], the UIO problem has been reformulated as an optimisation problem, and several evolutionary algorithms (EAs) such as genetic algorithm and simulated annealing [32, 50] were develop to tackle it. The experimental results show that EAs can efficiently find UIOs for some FSMs.

Theoretical investigations have confirmed that EAs can outperform random search on the UIO problem [75]. The expected running time of (1+1) EA on a counting FSM instance class is polynomial, while random search needs exponential time [75]. The UIO problem is NP-hard, so one can expect that there exist EA-hard instance classes. Theoretical results show that the EA configurations (operators and parameters) have an essential impact on finding the UIOs for an FSM efficiently [76].

The remainder of this section introduces definitions and notations of the UIO problem.

Definition 10 *(Finite State Machine).*

A finite state machine(FSM) is a quintuple: $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{X}, \boldsymbol{Y}, \delta, \lambda)$*, where* $\boldsymbol{X}, \boldsymbol{Y}$ *and* $\boldsymbol{S}$ *are finite and nonempty sets of input symbols, output symbols, and states, respectively;* $\delta : \boldsymbol{S} \times \boldsymbol{X} \longrightarrow \boldsymbol{S}$ *is the state transition function; and* $\lambda : \boldsymbol{S} \times \boldsymbol{X} \longrightarrow \boldsymbol{Y}$ *is the output function.*

Definition 11 *(Unique Input Output Sequence).*

A unique input output sequence for a state s_i *in an FSM is an input/output sequence* x/y*, where* $x \in \boldsymbol{X}^*, y \in \boldsymbol{Y}^*, \forall s_j \neq s_i, \lambda(s_i, x) \neq \lambda(s_j, x)$ *and* $\lambda(s_i, x) = y$*.*

Section 2 in [50] gives an example of the unique input output sequence problem.

To generate UIO using an EA, candidate solutions are represented by input strings restricted to $\mathbf{X}^n = \{0, 1\}^n$, where n is the number of states of the FSM. In general, the length of the shortest UIO is unknown, assume the objective is to search for a UIO of input string length n for state s_1 in all FSM instances. The fitness function is defined as a function of the state partition tree induced by the input sequence [50, 75, 76].

Definition 12 *(UIO fitness function).*

For a FSM M with m states, the fitness function $f : \boldsymbol{X}^n \longrightarrow \mathbb{N}$ is defined as $f(x) := m - \gamma_M(s,x)$, where s is the initial state, and $\gamma_M(s,x) := |\{t \in \boldsymbol{S} | \lambda(s,x) = \lambda(t,x)\}|$.

The instance size of the UIO problem is defined as the length of the input sequence n. The value of $\gamma_M(s,x)$ is the number of states in the leaf node of the state partition tree containing node s, and is in the interval from 1 to m. If the shortest UIO for state s in FSM M has length no more than n, then $f(x)$ has an optimum of $m-1$.

4.2.5.2 Target Algorithm

The $(\mu+\lambda)$ EAs described in Algorithm 3 is employed to tackle the UIO problem. The $(\mu+\lambda)$ EAs has three parameters: population size, variation operator, and selection operator, the domains of possible values for the three parameters are listed below:

- Population size: 3 different $(\mu+\lambda)$ options are considered: $\{(4+4),(7+3),(3+7)\}$.
- Variation operator $\mathcal{N}_j, (j = 1, 2, \dots, 12)$: Three variation operators with different probabilities are considered:
  - $\mathcal{N}_1(x) \sim \mathcal{N}_5(x)$: Bit-wise mutation, flip each bit with probability $p = c/n$, where $c \in \{0.5, 1, 2, n/2, n-1\}$;
  - $\mathcal{N}_6(x) \sim \mathcal{N}_9(x)$: flip c-th bits, where $c = \{1, 2, n/2, n-1\}$;

  - $\mathcal{N}_{10}(x) \sim \mathcal{N}_{12}(x)$: Non-uniform mutation[20], for each bit $i, 1 \leq i \leq n$, flip it with probability $\chi(i) = c/(i+1)$, where $c = \{0.5, 1, 2\}$.

  In total 12 variation operators are considered. These variation operators are applied to generate 12 fitness-probability clouds and obtain a set of 12 *aep* measures for each UIO instance.

- Selection operator $\mathcal{S}_i, (i = 1, 2)$: Two selection operators are considered:
  - Truncation Selection: Sort all $\mu + \lambda$ individuals in $\mathbf{P}^{(k)}$ and $\mathbf{P}_m^{(k)}$ by their fitness values, then select μ best individuals as the next generation $\mathbf{P}^{(k+1)}$.
  - Roulette Wheel Selection: Retain all the best individuals in $\mathbf{P}^{(k)}$ and $\mathbf{P}_m^{(k)}$ directly, and the rest of the individuals of the population are selected by roulette wheel.

Algorithm 3: $(\mu + \lambda)$- EA

```
Choose $\mu$ initial solutions $\mathbf{P}^{(0)} = \{x_1^{(0)}, x_2^{(0)}, \ldots, x_\mu^{(0)}\}$ uniformly at random
from $\{0,1\}^n$
$k \longleftarrow 0$
while Termination criterion not satisfied do
    $\mathbf{P}_m^{(k)} \longleftarrow \mathcal{N}_j(\mathbf{P}^{(k)})$          %%mutation
    $\mathbf{P}^{(k+1)} \longleftarrow \mathcal{S}_i(\mathbf{P}^{(k)}, \mathbf{P}_m^{(k)})$     %%selection
    $k \longleftarrow k + 1$
end
```

4.2.5.3 Performance Metric

The performance metric H and the performance evaluation function F to evaluate the performance need to be clearly defined. The former is to evaluate the performance of the algorithm configuration method over an entire set of testing instances, the later is to evaluate the performance of the target algorithm when executed under configuration θ on a problem instance *inst*.

Definition 13 *(Performance metric H and Performance evaluation function F)*

Let inst be a UIO instance, given a cut-off time t, F is defined as the number of function evaluations taken by the target algorithm executed under configuration θ to find a unique input output sequence for inst. H is defined as the mean value of F over an entire set of testing UIO instances. The lower the value of H, the better the algorithm configuration method.

4.2.5.4 Training Data Generation

The SVM takes as input a set of training samples $\{D_1, \cdots, D_m\}$, and their labels $\{L_1, \cdots, L_m\}$ where $L_i \in \{1, -1\}$.

A training sample $\mathbf{D_i} = (\mathbf{PF}, \theta)$, where **PF** in **D** denotes a problem instance, which is represented by a set of *aep* measures computed based on the fitness-probability clouds generated using different neighbourhood operators. In this case, a UIO instance is represented by a set of 12 *aep* measures $\{aep_1, \cdots, aep_{12}\}$ computed based on the fitness-probability clouds generated using 12 different neighbourhood operators. With respect to θ in **D**, θ represents a configuration of the target algorithm $(\mu + \lambda)$ EA, where the candidate configurations of the $(\mu + \lambda)$ EA are listed in the above section.

The label $\mathbf{L_i}$ for a training sample $\mathbf{D_i}$ is determined by the performance of the target algorithm $(\mu + \lambda)$ EA executed under configuration θ on the UIO instance. 100 independent runs are performed and the performance is measured by the performance metric F, which is the mean number of function evaluations taken by the target algorithm to find a unique input output sequence for the UIO instance. The performance metric F is then compared to a pre-defined threshold to determine the label $\mathbf{L_i}$ in the following way: if $F <$ threshold, $\mathbf{L_i} = 1$, otherwise $\mathbf{L_i} = -1$.

4.2.6 Experimental Studies

This section presents a detailed experimental study to evaluate the performance of the proposed automatic algorithm configuration method on 24 UIO instances with problem size $n = 20$ generated at random. First, the generalisation performance of the proposed automatic algorithm configuration method is evaluated by predicting whether a configuration of the $(\mu + \lambda)$ EA is suitable for solving a UIO instance. To ensure unbiased evaluation, the 10×10-fold cross validation [52] is used. Second, the performance of the proposed automatic algorithm configuration method is evaluated by comparing with ParamILS [66] which is a state-of-the-art automatic algorithm configuration method.

4.2.6.1 Generalisation Performance

Performance Measure

To evaluate the generalisation performance of the proposed classification method in predicting the performance of the target algorithm executed under configuration θ for solving a UIO instance, the most common performance measure of overall performance in binary classification is employed. The performance measure *accuracy* is the proportion of true results (correctly predicted performance of algorithm configurations on problem instances) among all predications.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4.4}$$

where TP, TN, FP and FN are the numbers of the true positive, the true negative, the false positive and the false negative, respectively.

10-fold cross-validation

Cross-validation is an estimator used to assess the performance of learning algorithm (SVM) and can be used to estimate the accuracy of a classifier [4]. 10-fold cross-validation partitions the original sample into 10 equal size subsamples. Of the 10 subsamples, a single subsample is retained as for testing, and the remaining k-1 subsamples are used for training. The cross-validation process is repeated 10 times (the folds). The classification accuracy of the classification algorithm is obtained by averaging the results from the folds. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Classification Accuracy

The measure-based classification is the most important part of the proposed automatic algorithm configuration method. The performance of the classification algorithm is illustrated by the experiments on predicting whether a configuration of the $(\mu + \lambda)$ EAs is effective for solving a UIO instance or not. The effectiveness of the configuration is determined by the performance measure of the target algorithm and a pre-defined threshold. In this case, the performance of the target algorithm is measured by the number of function evaluation taken to find a unique input output sequence for a UIO instance, and the threshold v is determined by a function $v = pr \cdot \bar{\mathbf{E}}_i$, where $\bar{\mathbf{E}}_i$ is the average from the performance $\mathbf{F}$ of each candidate configuration on the UIO instance i, and pr is a coefficient to adjust the value of v.

The 10-fold cross-validation was used to ensure unbiased evaluation. The performance of the measure-based classification was measured by the *accuracy* as defined in Equation 4.4. The overall performance of the measure-based classification is summarised in Table 4.1. The *accuracy* is an average from each of the 10-fold results. The variation of *accuracy* as the value of pr changes is illustrated in Figure 4.2.

As indicated in Table 4.1, pr controls the threshold v which is used to determine whether a configuration of $(\mu + \lambda)$ EA is deemed as suitable for solving a UIO instance or not. The second column in Table 4.1 showed the number of samples with label = 1, i,e., the configuration deemed as effective for solving the UIO instance. By varying the value of *pr* from 0.7 to 0.01, the *accuracy* of the measure-based classification varied in the range of 0.5 and 0.933 showing certain reliability. In particular, when *pr* was between 0.12 and 0.08, i.e., only configurations with performance significantly better than average are considered as effective, the measure-based classification demonstrated high reliability with *accuracy* = 0.864 at minimum. This result showed that the proposed measure-based classification approach can reliably determine if a configuration of the $(\mu + \lambda)$ EA is suitable for solving a given UIO instance, if a configuration is considered as effective when the $(\mu + \lambda)$ EA executed under the configuration shows high (significantly above average) performance.

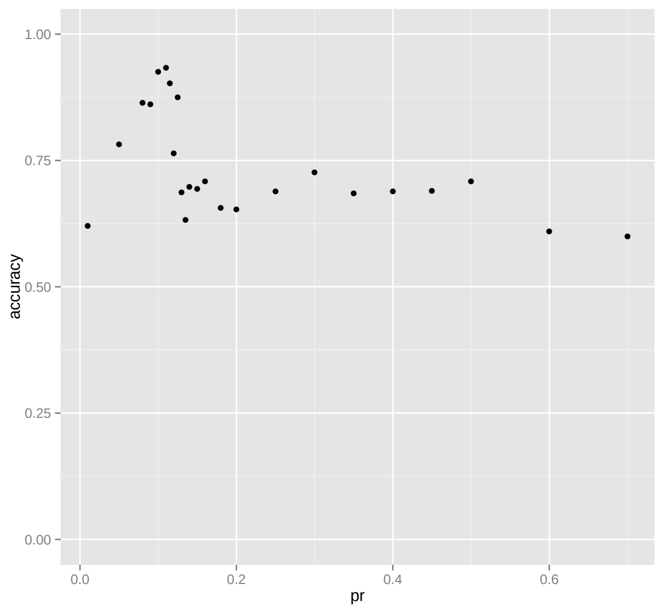


Figure 4.2: Plot of *accuracy* of the Classification Algorithm by Varying the Value of *pr* from 0.01 to 0.7.

4.2.6.2 Comparison with ParamILS

ParamILS is a state-of-the-art method for automatic algorithm configuration developed by Hutter et al. [66]. ParamILS has a large number of academic and industrial applications, which has yielded substantial improvements of heuristic al-

Table 4.1: Performance of the Measure-based Classification. *pr* is the Coefficient Used to Adjust the Threshold Which Determines Whether the Performance of the Target Algorithm is Considered Effective or Not. The 2nd Column Shows the Number of Positive Samples in the Training Data Set. *accuracy* is the Proportion of Accurate Predictions in Testing. The 3rd Column Gives the Mean and the Standard Deviation of *accuracy* from the 10-fold Results [78, 81].

pr	no. of samples label = 1	*accuracy*
0.7	1180	0.600 ± 0.0382
0.6	1115	0.610 ± 0.0288
0.5	1007	0.709 ± 0.0369
0.45	955	0.690 ± 0.0456
0.4	874	0.689 ± 0.0562
0.35	806	0.685 ± 0.0427
0.3	716	0.726 ± 0.0657
0.25	604	0.689 ± 0.0308
0.2	489	0.653 ± 0.0557
0.18	441	0.656 ± 0.0595
0.16	391	0.709 ± 0.0398
0.15	377	0.694 ± 0.0553
0.14	343	0.698 ± 0.0466
0.135	328	0.632 ± 0.0367
0.13	326	0.687 ± 0.0443
0.125	306	0.875 ± 0.0518
0.12	299	0.764 ± 0.0382
0.115	286	0.903 ± 0.0551
0.11	267	0.933 ± 0.0459
0.1	237	0.925 ± 0.0533
0.09	200	0.861 ± 0.0298
0.08	177	0.864 ± 0.0358
0.05	71	0.782 ± 0.0477
0.01	50	0.620 ± 0.0288

gorithms for hard combinatorial problems, such as propositional satisfiability (SAT), mixed integer programming (MIP), AI planning, answer set programming (ASP), and timetabling.

ParamILS performs one-size-fits-all algorithm configuration which is for finding an optimal configuration of the target algorithm for an entire set of instances. ParamILS is a direct search method which searches the configuration space using an iterative local search method (ILS). It uses a combination of default and random settings for initialisation, employs iterative first improvement as a subsidiary local search procedure, uses a fixed number (s) of random moves for perturbation, and always accepts better or equally-good configurations, but re-initialises the search at random with a small probability. Furthermore, it is based on a one-exchange neighbourhood, that is, it always considers changing only one parameter at a time. ParamILS returns a one-size-fits-all configuration for an entire set of problem instances.

We evaluated the performance of the proposed automatic algorithm configuration method, the problem difficulty measure-based classification (PDMC), against the ParamILS. ParamILS returned a one-size-fits-all configuration of the $(\mu + \lambda)$ EA for the entire set of 24 UIO instances. In contrast, PDMC returned a configuration of the $(\mu + \lambda)$ EA on a per-instance base. In particular, PDMC can indicate more than one configuration as effective for solving a UIO instance. When this is the case, PDMC returned one configuration selected uniformly at random from the set of configurations indicated as effective. The performance of PDMC and ParamILS were measured by the performance metric H described in Definition 13, i.e. the average number of function evaluations taken by the $(\mu + \lambda)$ EAs to solve the entire set of 24 UIO instances. The parameter pr was set at 0.1, i.e., a configuration with the number of function evaluations taken by the $(\mu + \lambda)$ EA for solving a UIO instance i below $0.1 \times \bar{\mathbf{E}}_i$ was deemed as effective. The results of PDMC was an average from each of the 10-fold results, since the 10 folds are partitioned randomly, PDMC has been executed 10 times. Similarly, the results of ParamILS was an average from 10 times of executions.

The experimental results of PDMC and ParamILS are summarised in Table 4.2 and illustrated in Figure 4.3. A statistical analysis has been performed to compare the significance of the results using the Wilcoxon rank-sum test [131], where p-value below 0.05 is considered to be statistically significant (confidence level 95%).

Table 4.2: Performance of ParamILS and PDMC Measured by the Average Number of Function Evaluations Taken by the $(\mu + \lambda)$ EAs in Solving an Entire Set of 24 UIO Instances, Where Mean and Standard Deviation of 10 Independent Executions are Given. The Best Average Result is Highlighted in Bold.

ParamILS	PDMC
1860.64 ± 11.09	**1630.21 ± 20.10**

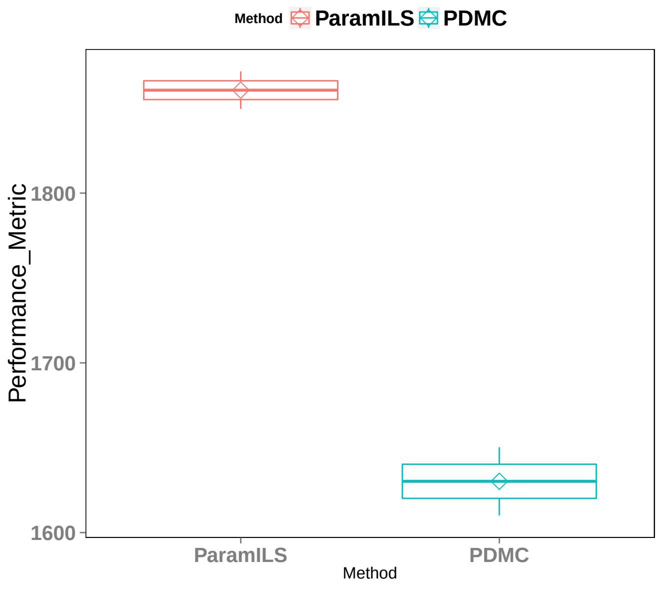


Figure 4.3: Illustration of Performance Between ParamILS and PDMC

The results showed that PDMC statistically significantly outperformed ParamILS in automatically configuring the $(\mu + \lambda)$ EAs for solving the UIO problem, and the average improvement of using PDMC over ParamILS was 12.3%.

4.3 Applying Elementary Landscape Analysis for Designing Novel Algorithm Configurations

4.3.1 Motivation

Finding good configurations for metaheuristics are essential to obtain robust and high algorithm performance. Whilst automatic algorithm configuration methods mainly concern selecting the best suited configuration from the configuration space, it is noted that the configuration space is not exhaustive and it is always useful to produce a novel, problem-specific configuration such as a new search operator or heuristic that potentially outperforms other configurations on a particular class of problem instances. The design of such novel and effective configurations must account for the characteristics of problem instances. This demand can be met by fitness landscape analysis, which is a powerful analytical tool, particularly in gaining in-depth understanding of the problem characteristics and the associated behaviours of algorithms.

This chapter for the first time bridges the gap between fitness landscape analysis and the design of novel heuristics, through proposing an approach to perform elementary landscape analysis and further explicitly apply the analytical results to design novel and effective local search heuristics.

In the next sections, a particular class of fitness landscapes called elementary landscape is introduced with emphasis on its unique properties which can potentially be applied to facilitate search, followed by descriptions of the proposed approach to perform elementary landscape analysis. Finally the proposed approach is applied to develop a novel and effective stochastic hill climbing heuristic for solving the next release problem in software engineering.

4.3.2 Elementary Landscape

Based on Grover's observations that the landscapes of certain combinatorial optimisation problems such as the Travelling Salesman Problem (TSP), could be characterised by a wave equation [49], Stadler [109] formally defines this kind of landscape as "elementary landscapes", where the objective function is an eigenfunction of the Laplacian of the graph induced by the neighbourhood operator.

Elementary landscapes possess unique properties which are considered beneficial for the search. In general, the properties are classified into two main categories. Implicit properties, landscapes with this property tend to be relatively smooth when contrasted to other combinatorial optimisation problems with well-studied local move operators, which could be considered to be an advantage for local search algorithms [130]. And the wave equation also imposes constraints on the structure of local optima and precludes the existence of certain plateau structures. Explicit properties, a wave equation in terms of the expected value of the neighbours is proposed, which more concretely expresses the properties of elementary landscapes [130]. Suppose x is some fixed but arbitrary element of X, y is an element drawn uniformly at random from the neighbourhood set $N(x)$ of x and $\overline{f}$ is the mean value over all solutions in X. On an elementary landscape, the following wave equation holds.

$$E[f(y)] = f(x) + \frac{k}{d}(\overline{f} - f(x))$$

for k which is fixed for the entire landscape. Since y is drawn uniformly at random, the expected value of the fitness value of a neighbour y is always equal to the average fitness value over all solutions in the neighbourhood [130]. Also all local minima lie below the average function value of the search space. In addition the expected fitness value of the full neighbourhood can be predicted by the wave equation. It is even possible to expand a partial neighbourhood during the search, and predict for the remaining neighbourhood. This property gives significant insight knowledge to the search algorithm that could be explicitly applied in designing algorithms.

For determining whether a regular landscape is elementary or not, Whitley et al. [130] constructed a component-based model that can be used to characterise a neighbourhood structure. In this model, the neighbourhood size is regular and denoted by d. The model consists of the following equations.

$$\overline{f} = p3 \sum_{c \in C} c$$

$$\begin{aligned} E\{f(y)\} &= f(x) - p1 f(x) + p2(\sum_{c \in C} c - f(x)) \\ &= f(x) - p1 f(x) + p2(\frac{1}{p3}\overline{f} - f(x)) \end{aligned}$$

where $0 < p1 < 1$ is the proportion of the intracomponents that that are removed from the solution in one move, $0 < p2 < 1$ is the proportion of the intercomponents that are added to the solution in a move. Finally, $0 < p3 < 1$ is the proportion of the total components in C that contribute to the cost function for any randomly chosen solution, which is independent of the neighbourhood size. Then the component theorem described in Theorem 1 (in Section 2.2) can be used to determine if a regular landscape is elementary or not.

4.3.3 Elementary Landscape Analysis

The special properties of elementary landscape, such as computing the average fitness value of the neighbourhood of any solution without evaluating all the solutions in the neighbourhood, shows a promising prospect of application in developing novel and effective local search heuristics. Nevertheless, to use the properties of elementary landscape requires that the fitness function is an eigenfunction of the Laplacian that describes the neighbourhood structure of the search space. This condition does not hold for an arbitrary fitness landscape and it is essential to find out whether a landscape is elementary or not, and further how to construct an elementary landscape by choosing an appropriate neighbourhood operator, in order to make use of the properties of elementary landscape.

Algorithm 4: Elementary Landscape Analysis

```
Input:
$f$: the fitness function of a combinatorial optimisation problem;
$N$: a set of neighbourhood operators;
Output:
A novel and effective local search heuristic;
begin
    Check to see if the fitness function $f$ is linearly decomposable;
    if $f$ is linearly decomposable then
        for neighbourhood operator $n \in N$ do
            Determine if the landscape $L$ formed by $n$ and $f$ is elementary
            using the component theorem;
            if $L$ is elementary then
                break;
            end
        end
        Apply the elementary properties of $L$ to develop a novel and effective
        local search heuristic for solving $f$;
    end
end
```

This chapter addresses this need by proposing an approach to perform elementary landscape analysis, which constructs an elementary landscape by choosing an appropriate neighbourhood operator for a fitness function, based on the component theorem [129] which determines whether a fitness function and a neighbourhood operator form an elementary landscape. It is worth noting the component theorem assumes a linearly decomposable fitness function. Once a landscape is proved to be elementary, the properties of elementary landscape can be used to develop novel and effective local search heuristics. The overall steps of elementary landscape analysis, i.e., constructing an elementary landscape and applying the results to develop novel heuristics, are summarised in Algorithm 4.

4.3.4 Applying Elementary Landscape Analysis to Develop Novel and Effective Stochastic Hill Climbing for the Next Release Problem

To demonstrate the effectiveness of the proposed approach, the elementary landscape analysis is applied to the next release problem (NRP) for developing novel and effective stochastic hill climbing heuristics. First, the next release problem is formally formulated as an optimisation problem. Second, elementary landscape analysis is applied to construct an elementary landscape based on the NRP, and the properties of elementary landscape are incorporated to develop novel and effective stochastic hill climbing for solving the NRP.

4.3.4.1 The Next Release Problem

The Next Release Problem (NRP) was originally formulated by Bagnall et al. [8]. The variant of the NRP studied in this chapter is a representative of a class of combinatorial optimisation problems where the fitness functions could be linearly decomposed. The problem is formulated as follows.

Given a software product, let R denote a set of candidate requirements to be considered to implement for the next release of the software, each $r \in R$ has an associated cost(r) which is a measure of the resource consumption to implement it.

and a weight w_i which reflects the requirement's importance. Also there is a budget for the total cost of the implemented requirements.

Associated with R, there is a directed acyclic graph G = (R,E) where $(r_i, r_j) \in E$ iff r_i is a prerequisite of r_j, G is also transitive since $(r_i, r_j) \in E \wedge (r_j, r_k) \in E \Rightarrow (r_i, r_k) \in E$. If the company decides to satisfy requirement r_i, it must satisfy the prerequisites of r_i. In a special case where no requirement has any prerequisite $E = \emptyset$, we say the problem is basic.

Assuming there are n requirements, the problem is to find a subset S with cardinality k of R, such that

$$\sum_{r_i \in S} w_i \text{ is maximised,}$$

$$\sum_{r_i \in S} cost(r_i) \text{ is minimised.}$$

Different search algorithms have been applied to NRP [8], but they were all experimental work. There is no analysis of whether the obtained results are good and whether they could be improved. There is no analysis either what characteristics the NRP has and whether the search algorithms used are appropriate.

4.3.4.2 Elementary Landscape Analysis for NRP

In most search algorithms, there is an objective function to guide the search. According to the problem formulation of the Next release problem (NRP), the objective function $f(S)$ is defined as [8]:

$$f(S) = \sum_{R_i \in S} w_i + Budget - \sum_{R_i \in S} cost(R_i) \tag{4.5}$$

The neighbourhood operator considered is the $two-exchange$, which randomly exchanges two requirements in S and $R \backslash S$.

We apply the component theorem to determine whether the objective function $f(S)$ and the neighbourhood operator *two−exchange* form an elementary landscape.

First of all, the objective function $f(S)$ is a combination of weights and costs, which is similar to Whitley's observation of components. Let the set $w_i - cost(R_i)$ make up the set of components C, where $|C| = |R|$, we could apply the component-based model and component theorem to determine whether the induced landscape is elementary or not.

According to the component based model (in Section 4.3), $p3$ and $\overline{f}$ are computed, since a solution consists of k items, the objective function $f(S)$ consists of k components, $p3 = \frac{k}{|R|}$ and $\overline{f}$ can be obtained:

$$\begin{aligned} \overline{f} &= p3 \sum_{c \in C} c \\ &= \frac{k}{|R|} (\sum_{R_i \in S} w_i - \sum_{R_i \in S} cost(R_i) + |R| * Budget) \end{aligned} \tag{4.6}$$

To compute $p1$, note that there are k components in any solution, and two-exchange changes exactly 2 components at a time, $p1 = 2/k$.

To compute $p2$, note that there are $|R| - k$ components with the components in $f(x)$ removed and 2 new components are picked from these, $p2 = \frac{2}{|R| - k}$.

Adding the terms to the component based model yields:

$$\begin{aligned} Avg\{f(y)\} &= f(x) - p1 f(x) + p2(\frac{1}{p3}\overline{f} - f(x)) \\ &= f(x) - \frac{2}{k} f(x) + \frac{2}{|R| - k}(\frac{|R|}{k}\overline{f} - f(x)) \\ &= f(x) + \frac{2|R|}{(|R| - k)k}(\overline{f} - f(x)) \end{aligned} \tag{4.7}$$

where $k = 2|R|$ and the neighborhood size is $d = (|R| - k)k$, and the landscape is elementary.

4.3.4.3 Novel and Effective Local Search Heuristic Using Properties of Elementary Landscape

It has been shown that the next release problem under the two-exchange neighbourhood has the search space that corresponds to elementary landscape. Further to implicit properties of elementary landscape, it is now possible to explicitly incorporate properties of elementary landscape to develop novel and effective local search heuristics.

First of all the stochastic hill climbing is considered. As a simple but effective local search algorithm, and involving only a single variation operator, deterministic hill climbing algorithms such as Simple Hill Climbing (first-best neighbour), Steepest-Ascent Hill Climbing (best neighbour) are extensively used in the field of optimisation. With all the advantages, deterministic hill climbing is by no means without its limitations, e.g., it is likely to get stuck in local optima due to their greedy acceptance of better neighbouring solutions. The strategy of the stochastic hill climbing algorithm (SHC) is proposed to address this limitation, which iterates the process of randomly selecting a neighbour for the incumbent solution and only accept it if it results in an improvement [37, 124]. A basic form of stochastic hill climbing is described in Algorithm 5.

Even though the SHC uses a stochastic process, it can still get stuck in local optima. To avoid getting stuck in local optima in optimising multi-modal functions, we chose to continue the search by accepting the sampled solution irrespective of whether it is better than the incumbent solution or not. The algorithm terminates after a fixed number of iterations. This iterative stochastic hill climbing for NRP is described in Algorithm 6.

Algorithm 5: Basic Stochastic Hill Climbing

```
begin
    $Current \leftarrow RandomSolution()$
    while $step < MaxStep$ do
        $Candidate \leftarrow RandomNeighbour(Current)$
        if $Candidate$ is better than $Current$ then
            $Current \leftarrow Candidate$;
        end
        $step++$;
    end
end
```

Algorithm 6: Iterative Stochastic Hill Climbing for NRP

```
begin
    $N = MaxStep$;
    $Current \leftarrow RandomSolution()$;
    $Best \leftarrow Current$;
    while Termination criteria not satisfied do
        $step = 0$;
        for $step < N$ do
            $step++$;
            $Candidate \leftarrow TwoExchange(Current)$
            if $f(Candidate) > f(Current)$ then
                $Current \leftarrow Candidate$;
            end
        end
        if $f(Current) > f(Best)$ then
            $Best \leftarrow Current$;
        end
        else
            $Current \leftarrow Candidate$;
        end
    end
end
```

The iterative stochastic hill climbing for NRP summarised in 5 is considered as the initial algorithm, the properties of elementary landscape are incorporated to improve the initial algorithm.

The properties of elementary landscape are categorised into two classes. One is implicit, which are inherent given the landscape is elementary and does not affect the design of the algorithm, e.g. relative smoothness. The other one is explicit, which can be explicitly applied to algorithm design, for example, allowing prediction for partial neighbourhoods.

Since the landscape is elementary, the algorithm can make use of the information provided by the properties of elementary landscape. In particular, the initial stochastic hill climbing does not have any knowledge about the neighbourhood of the incumbent solution. There is a lack of exploration in terms of the search, as the algorithm might keep exploiting a less promising neighbourhood and wastes a lot of efforts.

Nevertheless, for any solution x in an elementary landscape, one of the following observations is true:

- if $f(x) = \overline{f} \quad f(x) = E[f(y)] = \overline{f}$
- if $f(x) < \overline{f} \quad f(x) < E[f(y)] < \overline{f}$
- if $f(x) > \overline{f} \quad f(x) > E[f(y)] > \overline{f}$

When $f(x) < \overline{f} \quad f(x) < E[f(y)]$, the neighbourhood can be regarded as promising, the search is very likely to find an improving move by exploring this neighbourhood.

When $f(x) > \overline{f} \quad f(x) > E[f(y)]$, one cannot be sure that this neighbourhood is promising. However, in light of the elementary properties which allow prediction for partial neighbourhoods, the search can expand a partial neighbourhood of the incumbent solution x to discover promising neighbourhoods. The prediction is expected to consistently guide the search to more promising regions and reduce the efforts wasted in less promising regions.

By incorporating the insight knowledge of elementary landscape summarised above, a predictive stochastic hill climbing can be obtained. The developed elementary stochastic hill climbing is described in Algorithm 7.

4.3.4.4 Experimental Studies

Whilst some work formulated NRP as a multi-objective optimisation problem [36], this chapter considers NRP in its single-objective formulation, as this thesis is focussing on the single-objective optimisation. Several heuristic algorithms have been proposed to tackle the single-objective NRP, where a simulated annealing algorithm (SA) with Lundy and Mees cooling schedule ($\beta = 1\times 10^{-8}$) seemed to give the best overall results [8]. In Lundy Mees SA, the temperature drops after each move.

To evaluate the performance of the elementary stochastic hill climbing (ESHC) developed through elementary landscape analysis, ESHC is compared with the iterated stochastic hill climbing (ISHC) and the SA with Lundy and Mees cooling schedule. All three algorithms have been implemented in Matlab. The algorithm parameters used are presented in Table 4.3.

In total 16 NRP instances with different sizes of candidate requirements set $|R|$ and the selected requirements set $|S|$ are used in the experiments. These synthetic NRP instances are generated using the NRP data sets generator described in [36]. For each NRP instance, 100 independent runs of ESHC, ISHC and Lundy Mees SA are performed. The performance is measured by the quality of the best solution found within the cut-off time K, i.e., 1000 iterations. The cut-off time is determined by preliminary experiments which showed that the three algorithms would converge

Algorithm 7: Elementary Stochastic Hill Climbing

```
begin
    $C = \emptyset$;
    $Current = RandomSolution()$;
    while Termination criteria not satisfied do
        $Flag = False$;
            %% Current neighbourhood is promising
        if $f(Current) <= \overline{f}$ then
            for A steps, $A << N$ do
                $Candidate \leftarrow TwoExchange(Current)$;
                if $f(Candidate) > f(Current)$ then
                    $Current \leftarrow Candidate$;
                end
            end
        end
            %% Expand partial neighbourhood
        else
            for B steps, $B << N$ do
                $Candidate \leftarrow TwoExchange(Current)$;
                $C \cup Candidate$;
                Compute the expected fitness value of the neighbourhood
                $Exp[f(N(Candidate))]$;
                if $Exp[f(N(Candidate))] > f(Current)$ then
                    $Flag = True$;
                    $Break$;
                end
            end
                %% Partial neighbourhood is not promising
            if $Flag == False$ then
                $Current \leftarrow$ The best move in $C$
            end
                %% Partial neighbourhood is promising
            else
                for C steps, $C << N$ do
                    $Candidate \leftarrow TwoExchange(Current)$;
                    if $f(Candidate) > f(Current)$ then
                        $Current \leftarrow Candidate$;
                    end
                end
            end
        end
    end
end
```

within the cut-off time. The experimental results are illustrated in Table 4.4. The first column in Table 4.4 describes the parameters of the NRP instances: the value after slash is the size of candidate requirements set R and the ratio before slash specifies the proportion of R to be selected.

To conduct statistical comparisons of ESHC, ISHC and Lundy Mees SA over 16 NRP instances, the Friedman test is performed. The Friedman's test is a non-parametric equivalence of the repeated-measures analysis of variance (ANOVA) under the null hypothesis that all the algorithms are equivalent and so their ranks should be equal [40].

The Friedman's test is to determine whether there are global difference in the results. If the test result rejects the null hypothesis, i.e. these algorithms are equivalent, the post-hoc test can be applied to detect concrete differences among algorithms [42]. The power of the post-hoc test is much greater when all algorithms are compared with a control algorithm and not among themselves. Therefore the pairwise comparisons are not needed since the purpose of comparison here is to determine whether the proposed algorithm is better than the existing ones. In this case, ESHC is used as the control algorithm, where ISHC and Lundy Mees SA are compared against ESHC.

The Bonferroni-Dunn's test [34] is employed to conduct the post-hoc statistical analysis for detecting significant differences for the control algorithm ESHC. The performance of pairwise algorithms is significantly different if the difference between two average ranks is at least the critical difference:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{4.8}$$

where N denotes the number of instances, k denotes the number of algorithms and α denotes the level of significance considered. Here $N = 16$, $k = 3$, and $q_{0.05} = 2.241$.

The results from the Friedman's test and the post-hoc analysis are summarised in Table 4.5. First, given that the p-value of Friedman's test is lower than the level of significance considered $\alpha = 0.05$, there are significant differences among the results given by the three algorithms. Second, the differences between average ranks of ESHC vs. ISHC (1.75) and ESHC vs. Lundy Mees SA (1.1875) are greater than the critical difference $CD = 0.79$, it is concluded that ESHC significantly outperforms both ISHC and Lundy Mees SA.

Within the same cut-off time, i.e., 1000 iterations, the advantages of ESHC over ISHC and Lundy Mees SA showed that the predictive ability provided by the use of properties of elementary landscape in ESHC can be of help in determining if the current neighbourhood is promising or not, leading the search to focus on promising directions and avoid wasting efforts in unpromising neighbourhoods. In this way, the algorithm can achieve a balance between exploration and exploitation.

The use of the properties of elementary landscape in this case study is still preliminary, since only one elementary property has been explicitly incorporated, i.e., predicting the average fitness values of the partial or full neighbourhood. There is large room for developing approaches to make use of the properties of elementary landscape both theoretically and empirically. In particular, the decomposition of elementary landscape can generalise applications of results from elementary landscape analysis to an arbitrary combinatorial optimisation problem.

Table 4.3: Experimental Parameters

Parameters	Values
Cut-off time K	1000 iterations
Step size N in ISHC	10^{-3} * full neighbourhood size
Step size A in ESHC	0.1 * N
Step size B in ESHC	0.6 * A
Step size C in ESHC	0.4 * A
Control parameter β in Lundy Mees SA	1×10^{-8}

Table 4.4: Comparative Results of ESHC, ISHC and Lundy Mees SA on 16 NRP Instances, Where Mean and Standard Deviation of the Best Solutions Found by Three Algorithms Within the Cut-off Time are Given.

NRP Instance	ESHC	ISHC	Lundy Mees SA
PI-1 (10%/50)	31.2±0.88	25.8±2.17	29.8±2.1
PI-2 (20%/50)	63.2±1.4	61.9±1.9	61.5±0.5
PI-3 (50%/50)	121±0.5	118.6±1.56	120.1±1.1
PI-4 (80%/50)	146.2±2	146.1±2.74	146.1±1.13
PI-5 (10%/100)	74.76±0.43	72±1.02	72.3±0.98
PI-6 (20%/100)	139.1±1.5	137.9±0.81	137.8±0.3
PI-7 (50%/100)	272.4±0.95	270.3±0.79	270.9±0.9
PI-8 (80%/100)	327.1±1.1	327.1±1.1	328±0.1
PI-9 (10%/200)	159.9±0.9	150.5±0.71	152.1±0.37
PI-10 (20%/200)	243.1±0.7	241.7±0.5	241.9±0.2
PI-11 (50%/200)	539.6±0.1	536.8±0.5	537.1±1.4
PI-12 (80%/200)	373.4±0.64	371±0.71	371.5±0.38
PI-13 (10%/500)	362.7±0.57	358.5±0.71	359.7±2.6
PI-14 (20%/500)	677.2±0.84	675.4±5.37	674±2.5
PI-15 (50%/500)	1305.2±2.3	1302±0.45	1302.2±1.9
PI-16 (80%/500)	1610.8±4.2	1604.2±0.3	1605.1±1.2

Table 4.5: Results From the Friedman Test ($\alpha = 0.05$ and Average Ranks of ESHC, ISHC and Lundy Mees SA Over 16 NRP Instances, as Well as the Critical Difference $CD_{0.05}$ of the Bonferroni-Dunn's Test.

p-value of Friedman's Test	$CD_{0.05}$	ESHC	ISHC	Lundy Mees SA
< 0.0001	0.79	1	2.75	2.1875

4.4 Summary

In this chapter, the need and importance for finding suitable configurations of metaheuristics have been reiterated with emphasis on the need of a generic (problem-independent) approach for automatically configuring algorithms on a per-instance base. An interesting research question arises as to whether there are patterns or rules governing the choice of algorithm configurations, and whether such patterns can be learnt. This chapter provided an answer to this research question in two steps. First, instead of formulating the algorithm configuration as an optimisation problem, it has been reformulated as a decision problem. This reformulation motivates the development of a computationally inexpensive predictor of whether a configuration of a given algorithm is suitable for solving a particular problem instance. Second, fitness landscape analysis has been incorporated to build a generic approach which can perform automatic algorithm configuration on a per-instance base. The proposed approach is based on learning the pattern which governs the relationship between the algorithms configurations and the characteristics of problem instances, where the accumulated escape probability has been employed as features to characterise problem instances [78, 81]. In particular, results from the case study showed that the proposed approach can reliably determine if a configuration of the $(\mu+\lambda)$ EA is suitable for solving a UIO instance, provided a configuration was considered as effective when the $(\mu+\lambda)$ EA executed under the configuration showed high performance. Furthermore, the proposed approach significantly outperformed ParamILS, a state-of-the-art automatic algorithm configuration method, on automatically configuring the $(\mu+\lambda)$ EAs for solving the UIO problem.

Further to determining the best suited algorithm configuration within a finite set of existing algorithm configurations, it is always useful to produce a novel, problem-specific configuration (e.g. new heuristic/search operator) which potentially outperforms other configurations on a particular class of problem instances. This chapter has established a bridge connecting results of theoretical fitness landscape analysis and the design of novel, effective heuristics, through proposing an approach to perform elementary landscape analysis and further explicitly apply the analytical results to build novel and effective local search heuristics [80]. The results from the case study illustrated that the elementary hill climbing algorithm developed using results from elementary landscape analysis significantly outperformed both the standard stochastic hill climber and the state-of-the-art in solving the next release problem in software engineering.

Chapter 5

INCORPORATING FITNESS LANDSCAPE ANALYSIS FOR DYNAMIC ALGORITHM CONFIGURATION

5.1 Introduction

Finding good configurations of heuristic algorithms is essential to obtain robust and high algorithm performance. Traditionally algorithm configuration methods attempt to determine a priori the most appropriate configuration of a given heuristic algorithm for solving a particular problem instance. However, there is both empirical and theoretical evidence showing that the most effective configuration of a given algorithm for solving a particular problem instance can vary during the search process [116]. For example, theoretical analysis of mutation operators on binary encoded problems concluded that the mutation probability should be decreased as the genetic algorithm is approaching the global optimum [90]. Empirically Davis

[29] used a time-varying schedule of operator probabilities and observed improved performance.

This motivates the development of heuristic algorithms that dynamically adapt their configurations (search operators, numerical parameters, etc.) during the search process. These algorithms, often referred to as adaptive heuristic algorithms, aim to identify and select the most effective configuration of a given algorithm at each decision point during the search. In particular, the behaviours of such heuristic algorithms are adapted to specific characteristics of problem instances, and the generality of such algorithms would be improved to be able to solve a broad class of problems with diverse characteristics.

Adaptive heuristic algorithms are extensively studied in the literature with a number of approaches developed. The design of adaptive heuristic algorithms concerns two main issues, the credit assignment mechanism and the selection mechanism, where the former assigns a reward to a configuration based on evaluating the contribution of the configuration to the overall performance and the latter serves as a selection rule in charge of selecting the configuration to use.

The research focus of most existing approaches has been on the selection mechanisms, such as Probability Matching [24, 59, 116, 127], Adaptive Pursuit Method [115] and Multi-armed Bandit Paradigm [27]. However, these selection mechanisms are based on the mere observation for past behaviours of the candidate configurations. It is noted that considering only the past performance can be misleading on problems with complex structure such as deception [30, 88]. Furthermore, the optimal fitness assigned to a configuration is a dynamic random variable and the underlying distribution of this random variable changes as the search proceeds [29, 90].

To address this important issue and build enhanced adaptive heuristic algorithms, this chapter proposes to incorporate predictive information provided by the predictive problem difficulty measure to design a systematic mechanism for adaptive heuristic search. The predictive measure originates from fitness landscape analysis which explicitly quantifies the problem difficulty with respect to algorithms, based on the fitness function and the structure of search space [54]. Equivalently the measure can be used to quantify the predicted performance of a specific algorithm for solving a given problem instance.

A generic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure is proposed. The framework has access to a set of specific configurations for the target algorithm (the algorithm to be configured). At each decision point, the value of the predictive problem difficulty measure for the corresponding configuration is computed by exploiting information extracted from the search trajectory. It is pointed out that the predictive measure delivers the predicted performance of the target algorithm in candidate configurations, which enables selection of the best suited configuration to be based on the expected performance rather than the performance in the past. In the long run, a more accurate mapping from candidate configurations to search spaces with specific characteristics can be established. In terms of the selection mechanism, the proposed framework adopts the most widely used probability matching method, in which the predictive measures for candidate configurations are normalised into probabilities which represent the likelihoods to select those configurations at a decision point. The use of the predictive problem difficulty measure does not compromise the generality of the framework, as the measure does not require any a priori knowledge and is therefore suitable for cross-domain problems. To demonstrate the effectiveness of the proposed framework, it is applied to develop the fitness landscape based adaptive search algorithm (FAS) for the minimum vertex cover problem (MVC). FAS is tested on a large set of MVC instances with different characteristics.

This chapter contributes to the following:

a. Proposing a new approach to bridge the gap between fitness landscape analysis and algorithm design.

b. Proposing a generic framework for designing adaptive heuristic algorithms using predictive problem difficulty measures.

c. Designing an effective fitness landscape based adaptive search algorithm (FAS) for MVC by instantiating the proposed framework. FAS either outperforms or is comparable to the state-of-the-art algorithms for MVC on both widely studied benchmarks and real-world instances.

The remainder of this chapter is organised as follows. Section 5.2 introduces the predictive problem difficulty measure derived from our fitness landscape analysis, and the method to compute it. Section 5.3 describes a generic framework to design adaptive heuristic algorithms using the predictive problem difficulty measure. In Section 5.4, the proposed framework is applied to design an effective FAS for MVC. Section 5.5 briefly describes the experimental setup that is used throughout our experiments. Section 5.6 presents comparative results and analysis with state-of-the-art algorithms for MVC and the constituent algorithms. Further empirical analysis for FAS is also given. Finally, the chapter is summarised in Section 5.7.

5.2 Predictive Problem Difficulty Measure

It has been shown the notion of escape probability is critical in proving bounds of the expected runtime [125], which is usually taken as the difficulty measure in time complexity studies of metaheuristics [55, 57, 113]. Further investigation into the relationship between the escape probability and the expected time gives general lower bounds of the expected runtime in terms of the escape probability.

This section provides a brief introduction to the fitness-probability cloud [82] defined based on the notion of escape probability, and the predictive problem difficulty measure, the accumulated escape probability (aep) [82], derived from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms. In particular, since the original definition of the accumulated escape probability is general and has many degrees of freedom, a new definition of the accumulated escape probability (aep) is provided, which takes into account the arrival probabilities to differentiate contributions of escape probabilities at different fitness levels to the final aep measure. More importantly, an efficient methodology to compute the new aep is provided, which facilitates the application of aep in online optimisation techniques such as the adaptive heuristic algorithms.

5.2.1 Fitness-Probability Cloud and Accumulated Escape Probability

Escape Probability P_i^e, as defined in Definition 5 (in Section 3.3), represents the probability for a metaheuristic $\mathcal{A}$ to reach higher fitness levels from a certain fitness level. Based on the notion of escape probability, a natural way to study whether a problem is difficult or easy for a given algorithm, is to plot the fitness values against the escape probabilities, where the escape probability is obtained for each fitness level. Let f be a fitness function with m distinct fitness levels, the fitness-probability cloud (fpc), as in Definition 6 (in Section 3.3), is a set of points on a bi-dimensional plane:

$$fpc = \{(f_1, P_1^e), (f_2, P_2^e) \ldots, (f_m, P_m^e))\} \tag{5.1}$$

Although the fitness-probability cloud provides an illustrative characterisation of the underlying fitness landscape and gives an indication on the problem difficulty with respect to the algorithm applied to optimise it, the adaptive heuristic algorithm requires a quantitative evaluation of the performance of a candidate algorithm configuration. A problem difficulty measure, the accumulated escape probability (*aep*), is proposed which contains critical information on the fitness-probability cloud. The original definition of *aep* in Definition 7 (in Section 3.3) is quite general and has many degrees of freedom, which can be extended by determining a specific set of weights (w_i). Here the arrival probabilities (P_i^a) are considered as an appropriate set of weights, where P_i^a denotes the probability for the algorithm to reach fitness level i from other fitness levels. This is motivated by the fact that the escape probabilities at different fitness levels are not equally important, and the arrival probabilities can indicate the importance of corresponding escape probabilities in determining the problem difficulty in terms of *aep*. The particular *aep* is defined:

Definition 14 *(Accumulated Escape Probability)*

$$aep = \sum_{i=1}^{m} P_i^a P_i^e, \tag{5.2}$$

5.2.2 Efficient Method for Computing *aep*

The original method for computing *aep* via sampling is described in Chapter 3. In this section, an efficient method is presented for computing *aep* by exploiting the search trajectory instead of sampling from the search space.

We keep a record of a path of solutions in the search history of length K (K steps). By analysing these K samples, a set of distinct fitness values $\{f_1, \cdots, f_m\}$ can be obtained. The escape probability for the set of solutions with fitness f_i or level i is estimated as the number of improving steps (the step from level i to level j, where $f_j > f_i$) over the total number of steps from level i. The arrival probability of level i is estimated as the number of steps reaching level i over the total number of steps K. In the original method the Metropolis-Hastings sampling method [118] is used to give higher weights to more important search points, since not all fitness levels in the search space are equally important. Here the arrival probability is used for the same purpose. Hereinafter $\{f_1, \cdots, f_m\}$ are referred to as a set of fitness values obtained from a path of solutions in the search history of length K, and for each f_i, P_i^e is the estimated escape probability and P_i^a the estimated arrival probability from level i, respectively.

5.3 A Generic Framework for Designing Adaptive Heuristic Algorithms Using the Predictive Problem Difficulty Measure

In this section, a generic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure is presented. First, the motivations to introduce the predictive problem difficulty measure into the design of adaptive heuristic algorithms are explained. Second, the mechanism to incorporate the predictive measure with the adaptive heuristic algorithm is presented. Finally, the proposed framework is described in detail.

5.3.1 Motivations

A typical adaptive heuristic algorithm consists of two main components: a set of candidate configurations (heuristics/search operators) and a decision maker which determines which candidate configuration to use. The adaptive heuristic algorithm has access to a set of candidate configurations, and the goal of the decision maker is to combine them to produce an effective self-configured algorithm. The decision maker assesses the performance of each candidate configuration upon the progress of search, and employs a specific mechanism to select the optimal configuration of the algorithm at each decision point.

A number of adaptive mechanisms have been proposed in the literature, it is noted that most existing approaches are based on historical information only, mainly the quality of offspring produced. However, the goal of the adaptive heuristic algorithm is to find the best suited configuration to use when search proceeds to a new stage, rather than the configuration that performed best in the past. The predictive measure described in Section 5.2 quantifies the difficulty of a problem instance in relation to a specific algorithm, which can be used to approximately predict the performance of the algorithm on the problem instance. This generic problem difficulty measure can provide predictive information to enable the adaptive heuristic algorithm to select the configuration based on the expected performance instead of past performance only. As a result, a more accurate mapping between configurations of the target algorithm and search space with specific characteristics can be established.

5.3.2 Incorporating the Predictive Problem Difficulty Measure

In the literature there exist many interesting results on predictive fitness landscape measures, which provides an intuitive explanation of the problem hardness with respect to the heuristic/operator. Most applications of fitness landscape measures are offline, mainly involved in the algorithm selection stage, due to some limitations in the nature of measures or the methodology to compute them in practice. The fitness-distance correlation [70] requires known global optima, which is not possible for real-world problems. Others include negative slope coefficient (nsc) [119], correlation length and operator correlation [84], fitness variance [101], epistasis variance [28], all of them are computed based on a large set of samples. If these measures were to be used in an adaptive heuristic algorithm, sampling is required to generate sufficient samples at each decision point resulting in a huge computational overhead.

aep is a suitable measure to use in our proposed framework. The reasons are twofold. On one hand, *aep* is a reliable problem difficulty measure with theoretical underpinning in complexity theory of metaheuristics; on the other hand, the values of *aep* can be computed efficiently by exploiting the search trajectory upon the progress of search, which avoids incurring a considerable computational overhead since sampling is not required.

The adaptive heuristic algorithm using the predictive measure works in a way that, at each decision point, the algorithm passes the search information to the measure. Then, the measure for the most recent heuristic is updated and the probabilities for selecting the heuristics are updated and returned to the algorithm. The interactions between the adaptive heuristic algorithm and the predictive measure is illustrated in Figure 5.1.

5.3.3 The Proposed Framework

After describing how to incorporate the predictive problem difficulty measure into an adaptive heuristic algorithms, this subsection presents the algorithmic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure. The framework is summarised by the pseudo code in Algorithm 8.

The framework takes two inputs: the problem instance to solve and a set of problem specific configurations of the target algorithm. The output is the best solution ever found upon satisfying the stopping criteria. At each decision point, the value

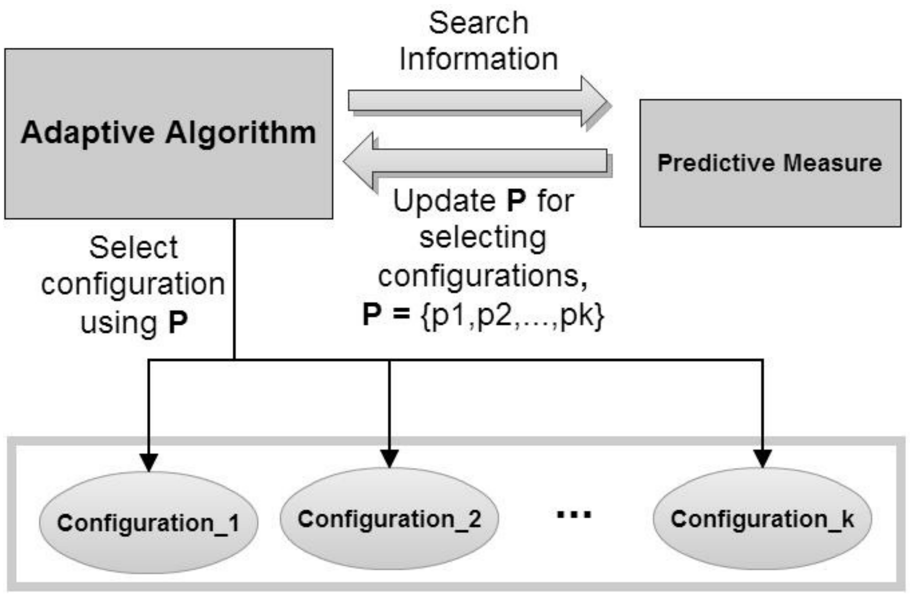


Figure 5.1: The Interactions Between the Adaptive Heuristic Algorithm and the Predictive Problem Difficulty Measure

of the predictive measure for the most recently applied configuration is computed based on the up-to-date search information. The algorithm is then to select the most effective algorithm configuration according to the probability distribution generated by the measure vector.

5.4 Fitness Landscape Based Adaptive Search Algorithm for the Minimum Vertex Cover

5.4.1 Minimum Vertex Cover

Given an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \cdots v_n\}$ is the set of vertices, E is the set of edges. The minimum vertex cover problem (MVC) is that of finding the minimum sized cover set $C \subseteq V$ such that for any edge $e\{u, v\} \in E$ at least one of its endpoints is in C.

MVC is an NP-hard [73] combinatorial optimisation problem with wide appli-

Algorithm 8: The Adaptive Algorithm Framework Using the Predictive Problem Difficulty Measure

```
$\{conf_1, conf_2, \cdots, conf_k\}$: A set of candidate configurations of the algorithm;
$\{m_1, m_2, \cdots, m_k\}$: Values of the predictive measure for corresponding candidates;
$P = \{p_1, p_2, \cdots, p_k\}$: Probabilities for selecting the heuristics;
begin
    Initialisation;
    while Termination criteria not satisfied do
        Configure the algorithm with the initial configuration $conf_{ini}$;
        Apply the algorithm;
        if Adaptive condition is satisfied then
            Compute the measure for the most recent configuration $conf_i$;
            Update $m_i$ in the measure vector;
            Normalise $\{m_1, m_2, \cdots, m_k\}$ to generate $P$;
            Select the new configuration according to $P$;
        end
    end
end
```

cations in network security, scheduling and computer graphics. It is worth noting that the Maximum Independent Set (MIS) problem and the Maximum Clique (MC) problem [13] with applications covering areas of information retrieval, experimental design, signal transmission, computer vision, and bioinformatics [69], are equivalent to MVC. Since MVC is NP-hard, one usually resorts to heuristics for solving it. A number of heuristic algorithms have been proposed, mainly local search algorithms.

Recent successful heuristics [10, 17, 18, 19, 48, 99, 104] mainly perform the plateau search on a single incumbent solution, i.e., swapping one vertex in the solution with another one not in the solution. In particular, many algorithms use the notion of edge weight, where a non-negative integer is assigned to an uncovered edge to diversify the search. This edge weighting scheme was first introduced in [104], then modified and applied subsequently [17, 18, 19]. For example, EWLS[17] and EWCC[18] only update the weights of uncovered edges upon reaching local optima, COVER[104] and NuMVC [19] update the weights of uncovered edges at each step, NuMVC even introduces a mechanism to reduce the edge weights upon reaching a pre-defined threshold.

5.4.2 Designing the Fitness Landscape Based Adaptive Search Algorithm for MVC

In this subsection, the proposed framework is employed to design a fitness landscape based adaptive search method (FAS) for tackling MVC. The FAS is summarised by the pseudo code in Algorithm 9. For solving MVC, the FAS takes two inputs, a set of heuristics for MVC and a pre-defined interval for reconfiguring the algorithm. The output is the best solution ever found upon satisfying the stopping criteria. In particular, the algorithm maintains a list of probabilities for selecting candidate heuristics. At each decision point, the value of the predictive measure for the most recently applied heuristic is computed by the method described in Section 5.2. The updated list of measures are normalised into probabilities for selecting heuristics. FAS constitutes its candidate heuristics set with a set of heuristics for MVC.

5.4.2.1 Constituent Heuristics

Typically heuristic algorithms solve MVC by iteratively solving the k-vertex problem. The initial cover set of size k is generated at random. Basically the algorithm maintains an incumbent solution of size k and iteratively performs plateau search which exchanges two vertices in and out of the solution until it becomes a vertex cover. The constituent heuristics of FAS are based on the plateau search and employ different strategies to select the vertex from the solution C and the vertex not from the solution $V\backslash C$ for exchange. For the sake of diversification, FAS maintains a tabu list of size one that prohibits the vertex swapped into the solution being swapped out immediately. The five heuristics in FAS are briefly described below and illustrated in Algorithm 10-14, with the notations explained in Table

Algorithm 9: Fitness Landscape Based Adaptive Search

```
$H\{heuristic_1, heuristic_2, \cdots, heuristic_k\}$: A set of candidate heuristics;
$M = \{m_1, m_2, \cdots, m_k\}$: Values of the predictive measure for the heuristics;
$P = \{p_1, p_2, \cdots, p_k\}$: Probabilities for selecting the heuristics;
$intv$: pre-defined intervals for updating operators/heuristics;
begin
    Generate an initial solution $ind$;
    Select the initial heuristic $heuristic_i$ uniformly at random;
    $step = 0$;
    while Termination criteria not satisfied do
        $ind$ = search $(ind, heuristic_i)$;
        $step++$;
        if $step == intv$ then
            Compute the predictive measure $m_i$ of $heuristic_i$;
            Update $m_i$ in $M$;
            Transform $M$ into the new probability distribution $P'$;
            $heuristic_i$ = heuristic selected from $H$ based on $P'$;
            $step = 0$;
        end
    end
end
```

5.1. Note that H0 is based on the edge weighting scheme adopted from NuMVC [19]. In H1-H4, only a subset of vertices in the solution are considered for swap, the definition of the subset is extended from the neighbourhood definition used by Pullan and Hoos [97].

Table 5.1: Notations Used in the Constituent Heuristics

Symbols	Meanings
$s_i = \{0,1\}$	state of vertex v_i, $s_i = 1$ if $v \in C$, $s_i = 0$ if $v \notin C$
$S = \{0,1\}^n$	S is a binary string which represents the candidate solution
$w(e)$	weight (non-negative integer) associated with edge e
$w(v)$	weight of vertex v, $w(v) = \sum_{u \notin C, e=\{u,v\}} w(e)$
$age(v)$	age of a vertex (steps since its state was last changed)
$D = (p_0, \cdots, p_{n-1})$	probabilistic model of solutions, $\forall i \in V$, p_i represents the likelihood of vertex i in C.
$N(i) = \{v_j \mid v_j \in V, \{v_i, v_j\} \in E\}$	the vertices adjacent to v_i
$S_p(C) = \{v_i \mid v_i \in C, \lvert V \backslash C \backslash N(i) \rvert = p\}, p = \{0,1\}$	the set of vertices adjacent to exactly p vertex in $V \backslash C$

- H0

- From C: Select a vertex $v \in C$ with the lowest $w(v)$, breaking ties in favour of the vertex with a larger $age(v)$.
- From $V \backslash C$: Select an uncovered edge e at random, and select the endpoint with a higher weight, ties broken favouring the vertex with a larger $age(v)$.

- H1
  - From C: First obtain $S_0(C)$ and $S_1(C)$, if $S_0(C)$ is not empty, randomly choose a vertex $v \in S_0(C)$; otherwise choose a vertex $v \in S_1(C)$ at random.
  - From $V \backslash C$: The adjacent vertex of v.
- H2
  - From C: First obtain $S_0(C)$ and $S_1(C)$, if $S_0(C)$ is not empty, randomly choose a vertex $v \in S_0(C)$; otherwise choose a vertex $v \in S_1(C)$ with the highest p_i within the probabilistic model D.
  - From $V \backslash C$: The adjacent vertex of v.
- H3
  - From C: First obtain $S_0(C)$ and $S_1(C)$, if $S_0(C)$ is not empty, randomly choose a vertex $v \in S_0(C)$; otherwise choose a vertex $v \in S_1(C)$ with the lowest p_i within the probabilistic model D.
  - From $V \backslash C$: The adjacent vertex of v.
- H4
  - From C: First obtain $S_0(C)$ and $S_1(C)$, if $S_0(C)$ is not empty, randomly choose a vertex $v \in S_0(C)$; otherwise choose a vertex $v \in S_1(C)$ at random.

– From $V \backslash C$: A randomly selected vertex.

Algorithm 10: Heuristic H0

```
begin
    select a vertex $v \in C$ with the lowest $w(v)$,
    ties broken favouring the vertex with greater $age(v)$;
    $C \leftarrow C \backslash \{v\}$;
    select an uncovered edge $e\{u, w\}$ at random,
    select the endpoint $w$ with greater weight;
    $C \leftarrow C \cup \{w\}$;
    for each uncovered edge $e$ do
        $w(e) \leftarrow w(e) + 1$;
    end
end
```

5.5 Experimental Setup

5.5.1 Benchmarks

In this section, an experimental evaluation is performed for the FAS on two standard benchmarks in the literature of MVC: DIMACS [69] and BHOSLIB [135], and a new set of instances: MESH [107]. DIMACS and BHOSLIB were widely used in the MVC research, whilst MESH was recently introduced which originated from a real-world application in Computer Graphics [5].

The DIMACS benchmarks originally come from the Second DIMACS Challenge Test Problems (1992-1993), which contain many instance families generated from different problems.

- *p_hat* family: a generalization of the classical uniform random graph generator, with a wide node degree range.
- *brock* family: random graphs with a unique minimum vertex cover, purposely developed to defeat greedy heuristics.

Algorithm 11: Heuristic H1

```
begin
    Compute $S_0(C)$, $S_1(C)$;
    if $S_0(C)$ is not empty then
        randomly choose a vertex $v \in S_0(C)$;
        $C \leftarrow C \backslash \{v\}$;
        return;
    end
    if $S_1(C)$ is not empty then
        randomly choose a vertex $v \in S_1(C)$;
        choose the only vertex $w \in V \backslash C$ adjacent with $v$;
        $C \leftarrow C \backslash \{v\} \cup \{w\}$;
        return;
    end
    else
        randomly choose a vertex $v \in C$;
        randomly choose a vertex $w \in V \backslash C$;
        $C \leftarrow C \backslash \{v\} \cup \{w\}$;
        for each vertex $v_i \in C$ do
            Increase $p_i$ in $D$;
            Update the probabilistic model $D$;
        end
        return;
    end
end
```

Algorithm 12: Heuristic H2

```
begin
    Compute $S_0(C)$, $S_1(C)$;
    if $S_0(C)$ is not empty then
        choose a vertex $v \in S_0(C)$ with the highest $p_i$ in the probabilistic
        model $D$,
        ties broken randomly;
        $C \leftarrow C \backslash \{v\}$;
        return;
    end
    if $S_1(C)$ is not empty then
        choose a vertex $v \in S_0(C)$ with the highest $p_i$ in the probabilistic
        model $D$,
        ties broken randomly;
        choose the only vertex $w \in V \backslash C$ adjacent with $v$;
        $C \leftarrow C \backslash \{v\} \cup \{w\}$;
        return;
    end
    else
        randomly choose a vertex $v \in C$;
        randomly choose a vertex $w \in V \backslash C$;
        $C \leftarrow C \backslash \{v\} \cup \{w\}$;
        for each vertex $v_i \in C$ do
            Increase $p_i$ in $D$;
            Update the probabilistic model $D$;
        end
        return;
    end
end
```

Algorithm 13: Heuristic H3

```
begin
    Compute $S_0(C)$, $S_1(C)$;
    if $S_0(C)$ is not empty then
        choose a vertex $v \in S_0(C)$ with the lowest $p_i$ in the probabilistic model
        $D$,
        ties broken randomly;
        $C \leftarrow C \backslash \{v\}$;
        return;
    end
    if $S_1(C)$ is not empty then
        choose a vertex $v \in S_0(C)$ with the lowest $p_i$ in the probabilistic model
        $D$,
        ties broken randomly;
        choose the only vertex $w \in V \backslash C$ adjacent with $v$;
        $C \leftarrow C \backslash \{v\} \cup \{w\}$;
        return;
    end
    else
        for each vertex $v_i \in C$ do
            Increase $p_i$ in $D$;
            Update the probabilistic model $D$;
        end
        randomly choose a vertex $v \in V$;
        $C \leftarrow \{v\}$;
        return;
    end
end
```

Algorithm 14: Heuristic H4

```
Heuristic H4
begin
    Compute $S_0(C)$, $S_1(C)$;
    if $S_0(C)$ is not empty then
        randomly choose a vertex $v \in S_0(C)$;
        $C \leftarrow C \backslash \{v\}$;
        return;
    end
    if $S_1(C)$ is not empty then
        randomly choose a vertex $v \in S_1(C)$;
        choose the only vertex $w \in V \backslash C$ connected with $v$;
        $C \leftarrow C \backslash \{v\} \cup \{w\}$;
        return;
    end
    else
        for each vertex $v_i \in C$ do
            Increase $p_i$ in $D$;
            Update the probabilistic model $D$;
        end
        randomly choose a vertex $v \in V$;
        $C \leftarrow \{v\}$;
        return;
    end
end
```

- *gen* family: random graphs with a unique minimum vertex cover.
- *hamming* family: from coding theory problems.
- C family: random graphs with given size n and density $0.p$.
- MANN family: Steiner triple graphs.
- *keller* family: based on Keller's conjecture on tilings using hypercubes.

The BHOSLIB (Benchmarks with Hidden Optimum Solutions) benchmarks are a set of transformed satisfiability instances arising from SAT'04 competition. These instances were randomly generated using the model RB in the phase transition area. Due to the fact that the optimal solutions were deliberately hidden, these instances have been proved to be hard both theoretically [136] and practically [135].

The last family, MESH, is relatively new in the context of MVC and MIS [5]. MESH is motivated by an application in computer graphics [107]. To be more specific, to process a triangulation efficiently in graphics hardware, the algorithm needs to find a small subset of triangles that covers all the edges in the mesh. This is equivalent to finding a small set cover on the corresponding dual graph (adjacent faces in the original mesh become adjacent vertices in the dual). The MESH family contains the duals of well-known triangular meshes. The vertices of degree one and zero from the dual have been eliminated during the conversion from the original primal meshes to the duals, since there is always a maximum independent set that contains them. Almost all vertices in the resulting MVC instances have degree three. The MESH instances originated from real-world problems, for which the optimal solutions are unknown. However, Sander et al. [107] have computed lower bounds on the cover sizes for several MESH instances, e.g., gargoyle (10880), feline (21937) and bunny (36382).

5.5.2 Performance Measures

On benchmark problem instances with known optimal solutions, comparing the performances of different algorithms implies to take into account that some algorithms may have a small probability of success but converge quickly whereas others may have a larger probability of success but are slower. Auger et al. [7] use the expected running time (ERT) and the expected running length (ERL) to measure the performance of an algorithm by conducting independent restarts of the algorithm. The ERT and ERL to reach the optimal solution are:

$$ERT(opt) = RT_S + \frac{1 - p_s}{p_s} RT_{US} \tag{5.3}$$

$$ERL(opt) = RL_S + \frac{1 - p_s}{p_s} RL_{US} \tag{5.4}$$

where p_s denotes the fraction of successful trials, the running times RT_S and RT_{US} denote the average cpu time for successful and unsuccessful trials, respectively. And the running length RL_S and RL_{US} denote the average number of function evaluations for successful and unsuccessful trials, respectively. ERT and ERL take into account both the success rate and time cost and give a fair evaluation on an algorithm's performance, therefore ERT and ERL are employed as the performance measure to compare the performance of different algorithms for solving the DIMACS and BHOSLIB benchmarks.

On real-world problem instances where the optimal solutions are unknown, the most common way of measuring the performance of an algorithm is in terms of the quality of the solution found in comparison to the quality of the solution found by some other algorithm. Given sufficient independent runs of both algorithms, it is reported whether there is a statistically significant difference in the quality of the solutions found by the algorithms. To ensure fair comparisons between different algorithms, the quality of the solution found is usually based on the best or average solution found after a set number of function evaluations by the algorithm.

On the MESH instances, the performance of different algorithms are measured by the average of the best solution found within a cut-off time over a sufficient number of independent runs.

5.5.3 Computing Environment

The FAS is implemented in C++. The source code of PLS and NuMVC are also implemented in C++ as provided by their authors. All three algorithms are compiled by g++ with the '-O2' option. All runs were made on a 3.4 GHz Intel CPU and 8GB RAM under Linux. To ensure fair comparison, for each algorithm, 100 independent runs are performed on the DIMACS and BHOSLIB benchmarks with different random seeds, and 20 independent runs on the MESH instances. The parameters used in the experiments are listed below:

- **Adaptive Interval**: The adaptive interval for FAS is set at 1e+07 steps, i.e., the algorithm updates the search heuristic every 1e+07 steps.
- **Stopping Criteria**: Since the optima of DIMACS and BHOSLIB benchmarks are known, for these instances each run terminates upon finding the optimal solution or reaching the cut-off condition which is set at 1.5e+09 steps. Since the optima of MESH instances are unknown, each run terminates upon reaching the cut-off condition which is set at 1.5e+09 steps.

5.6 Experimental Studies

This section presents a detailed experimental study to evaluate the performance of FAS on DIMACS, BHOSLIB and MESH instances. First, the performance of FAS is evaluated by comparing with two other state-of-the-art algorithms PLS and NuMVC. Second, the performance of FAS is evaluated by comparing with constituent algorithms including the individual heuristics H0 - H4 in FAS and the algorithm using random heuristic selection.

Furthermore, to gain a deep understanding of the behaviours of FAS, additional empirical analysis is performed. First, the run-time behaviours of FAS are explored through its run-time distributions (RTD). Then the impact of two parameters on the algorithm performance is studied. Finally the behaviours of both FAS and the predictive measure on different problem instances are visualised for investigation.

5.6.1 Comparison with State-of-the-art Algorithms

In this section, FAS is compared with PLS [99] and NuMVC [19], both acknowledged as state-of-the-art algorithms for MVC in the literature.

- **PLS**: Phased local search algorithm, originally proposed for the maximum clique problem, a later version for MVC is given in [96]. PLS demonstrates dominating performance on most DIMACS benchmarks except for the MANN family.
- **NuMVC**: A local search algorithm for MVC. NuMVC presents competitive results with PLS on DIMACS benchmarks except for the brock family, and it is the best algorithm on BHOSLIB benchmarks in the literature.

5.6.1.1 Comparative Results on DIMACS

Comparative Results on DIMACS benchmarks are reported in Table 5.2. Most DIMACS instances are now too easy for modern solvers, which can be easily solved by all three algorithm in a few seconds, the results on those instances are omitted here. The Friedman's test [40] is employed to determine if there are global difference in the results given by three algorithms, where p-value below 0.05 is considered to be statistically significant (confidence level 95%). The results suggest that there is no global difference between the three algorithms in terms of the success rate (p-value = 0.1561). Since NuMVC and PLS completely failed on some instances, it is not possible to conduct statistical analysis on performance measured by ERL.

Statistical analysis suggests that the three algorithms are equivalent over the entire set of DIMACS benchmarks, the Wilcoxon rank-sum test [131] with 95% of statistical confidence is employed to compare the results on each instance, in terms of 100 independent runs. As indicated in Table 5.2, on the brock instances which are explicitly generated to defeat greedy heuristics, NuMVC failed on its hard instances like most greedy heuristics, whilst both FAS and PLS achieve significantly better results, however, the statistical analysis shows that there is no significant difference between FAS and PLS. On the putatively hard MANN_a81 instance, most previous heuristics completely failed, so does PLS. FAS achieves a 97% success rate which is much better than 54% by NuMVC, and the statistical test shows that FAS significantly outperforms NuMVC in terms of the run length. There is no significant difference among the three algorithms in terms of both the success rate and the run length on C4000.5, C2000.9, gen400_p0.9_55, keller6 and p_hat1500-1.

In sum, FAS significantly outperforms both NuMVC and PLS on the putatively hard MANN_a81 instance, and is comparable to either PLS or NuMVC on the rest of DIMACS instances.

Table 5.2: Comparative Results on Selected DIMACS Benchmarks, the Best Success Rate Obtained for Each Instance is Highlighted in Bold.

Instance	n	*opt*	NuMVC			PLS			FAS		
			Success	ERL	ERT(s)	Success	ERL	ERT(s)	Success	ERL	ERT(s)
brock400_2	400	371	96	645631471	572.3	**100**	425333	0.44	**100**	15192380	11.31
brock400_4	400	367	100	6322882	4.98	100	61292	0.1	100	5201368	3.5
brock800_2	800	776	0	n/a	n/a	**100**	30410290	72.8	**100**	133592306	239.12
brock800_4	800	774	0	n/a	n/a	**100**	8337099	20.3	**100**	7775114	17.54
C2000.9	2000	1920	1	149002136870	268949.1	0	n/a	n/a	**3**	49348820324	86316.19
C4000.5	4000	3982	100	7989361	124.4	100	4927525	114.87	100	8098165	148.06
gen400_p0.9_55	400	345	100	38136	0.13	100	83928	0.18	100	38009	0.11
keller6	3361	3302	100	438972	28.19	100	3369403	18.17	100	462975	3.31
MANN_a45	1035	690	**100**	90427670	70.4	0	n/a	n/a	**100**	168461871	90.84
MANN_a81	3321	2221	54	1939835273	4800.59	0	n/a	n/a	**97**	477085773	999.76
p_hat1500-1	1500	1488	100	585133	5.01	100	194304	2.03	100	593697	4.01

5.6.1.2 Comparative Results on BHOSLIB

Comparative Results on BHOSLIB benchmarks are illustrated in Table 5.3. It is clear that PLS is dramatically worse than NuMVC and FAS on most instances except for the relatively simple ones. The Wilcoxon rank-sum test [131] with 95% of statistical confidence is used to analyse the results given by NuMVC and FAS. In particular, FAS finds the optimal solution with 100% success rate for 21 out of 30 instances. For NuMVC the number is 22, the only difference occurs on frb53-24-4 where the success rate of FAS is 97%. The statistical analysis shows that the differences between FAS and NuMVC are insignificant in terms of both the success rate and ERL, with p-values at 0.3412 and 0.2861, respectively.

In sum, FAS significantly outperforms PLS and is comparable to NuMVC on BHOSLIB benchmarks.

5.6.1.3 Comparative Results on MESH

Comparative Results on MESH are summarised in Table 5.4. It is worth noting that MESH instances are fundamentally different from the previous DIMACS and BHOSLIB benchmarks. On one hand, MESH instances are from real-world problems

Table 5.3: Comparative Results on the BHOSLIB Benchmarks, the Best Success Rate Obtained for Each Instance is Highlighted in Bold.

Instance	n	*opt*	NuMVC			PLS			FAS		
			Success	ERL	ERT(s)	Success	ERL	ERT(s)	Success	ERL	ERT(s)
frb40-19-1	760	720	100	213918	0.32	100	730969	0.81	100	209992	0.28
frb40-19-2	760	720	100	4256938	3.86	100	16584810	17.94	100	7208289	6.83
frb40-19-3	760	720	100	851291	0.88	100	4622106	5.08	100	846588	0.84
frb40-19-4	760	720	100	2852893	2.65	100	17570890	19.86	100	4687772	4.55
frb40-19-5	760	720	100	12201973	10.83	100	119752273	130.27	100	22820479	22.92
frb45-21-1	945	900	100	2463140	2.71	100	26285666	30.11	100	3180485	3.55
frb45-21-2	945	900	100	4506735	4.82	100	83355260	96.36	100	6405142	7.2
frb45-21-3	945	900	**100**	11841684	12.38	85	740500037	858.72	**100**	31652816	35.66
frb45-21-4	945	900	100	4266486	4.56	100	17859728	21.35	100	6436128	7.09
frb45-21-5	945	900	100	8949554	9.42	100	114747722	136.19	100	21764395	24.49
frb50-23-1	1150	1100	**100**	32154438	38.95	99	359320205	475.44	**100**	63777048	80.01
frb50-23-2	1150	1100	**100**	148966929	182.58	0	n/a	n/a	**100**	290146916	347.96
frb50-23-3	1150	1100	**98**	450069508	554.25	0	n/a	n/a	91	698235096	816.04
frb50-23-4	1150	1100	**100**	5028196	6.35	100	14916125	19.54	**100**	7002153	8.9
frb50-23-5	1150	1100	**100**	15781442	19.5	100	239489970	311.78	**100**	35739625	43.64
frb53-24-1	1272	1219	**85**	752108726	995.41	0	n/a	n/a	71	1226530252	1569.74
frb53-24-2	1272	1219	**100**	153216458	203.5	3	49130964367	67082.67	**100**	231949492	302.3
frb53-24-3	1272	1219	**100**	27031953	36.06	60	1345232923	1823.23	**100**	81763482	108
frb53-24-4	1272	1219	**100**	244370626	319.34	30	4276679947	5834.77	97	348219300	445.96
frb53-24-5	1272	1219	**100**	30242273	39.23	40	2458241640	3348.8	**100**	60860727	81.53
frb56-25-1	1400	1344	**96**	416907891	581.87	0	n/a	n/a	94	614172789	837.07
frb56-25-2	1400	1344	**91**	552882515	772.36	0	n/a	n/a	**91**	741094488	969.44
frb56-25-3	1400	1344	**100**	78477490	109.72	0	n/a	n/a	**100**	181148020	247.67
frb56-25-4	1400	1344	**100**	25231074	35.4	35	3264476237	4711.4	**100**	80152900	112.9
frb56-25-5	1400	1344	**100**	14444131	20.32	93	395447115	565.86	**100**	44546145	63.82
frb59-26-1	1534	1475	**83**	844792024	1271.41	0	n/a	n/a	77	1015432521	1478.78
frb59-26-2	1534	1475	**43**	2659650657	4167.1	0	n/a	n/a	25	5216501456	7621.81
frb59-26-3	1534	1475	**94**	471839485	727.53	20	6419008890	9516.55	87	743375266	1066.11
frb59-26-4	1534	1475	**86**	702870774	1080.79	0	n/a	n/a	75	1042081257	1480.42
frb59-26-5	1534	1475	**100**	35111811	53.84	89	538749308	781.47	**100**	64968815	95.5

and the optimal solutions are unknown. On the other hand, MESH instances are large, very sparse graphs with average vertex degree three. Here FAS has clear advantages over both PLS and NuMVC across all instances. No statistical analysis is required in this case, as for each MESH instance, the worst results given by FAS are better than the best results given by PLS and NuMVC.

Table 5.4: Comparative Results on MESH Instances, the Best Average Solution and the Best Solution Found in Each Case are Highlighted in Bold.

Instance	n	NuMVC			PLS			FAS		
		MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN
blob	16068	8846	8844.5	8839	8856	8850.1	8843	8825	**8822.1**	**8820**
gargoyle	20000	11198	11195.8	11193	11198	11188.2	11176	11152	**11149.1**	**11148**
face	22871	12704	12701.9	12700	12707	12701.1	12697	12658	**12656.1**	**12654**
feline	41262	22478	22474.3	22467	22564	22551.1	22541	22428	**22425.2**	**22419**
gameguy	42623	21915	21912.1	21910	22048	22030.2	22015	21904	**21901.3**	**21899**
bunny	68790	36508	36506.1	36505	36693	36667.2	36653	36490	**36480.7**	**36472**

5.6.2 Comparison with Constituent Algorithms

In this section, an empirical comparison and analysis between FAS and constituent algorithms is carried out to show the effectiveness of the proposed measure-driven adaptive mechanism.

The investigation is performed on representative instances selected from different instance classes. For DIMACS benchmarks, $brock800\_2$ and MANN_a81 are selected, as they are the hardest instance of *brock* and MANN family, respectively. For BHOSLIB benchmarks, $frb59$-26-1 and $frb59$-26-2 are selected, since they are of large size and challenging difficulty within the BHOSLIB benchmark. For MESH instances, $face$ and *gameguy* are selected. $face$ is of reasonable size within MESH instances, while *gameguy* is much larger and tends to be more difficult.

5.6.2.1 Comparison with Individual Heuristics

First of all, FAS is compared with the individual heuristics H0 - H4 in FAS.

The comparative results on selected instances from DIMACS and BHOSLIB benchmarks are summarised in Table 5.5. Since H0 adopts the strategies to select vertices for exchange from NuMVC [19], it gives exactly the same performance as NuMVC under identical experiment settings. It is clear that H0 and FAS are significantly better than H1-H4, as H1-H4 completely failed on the frb instances, and it has been shown that H0 and FAS are equivalent on frb instances. In particular, FAS significantly outperforms all individual heuristics on the putatively hard MANN_a81 instance. As for the brock800_2, the Wilcoxon rank-sum test [131] with 95% of statistical confidence has been applied which indicates that H3, H4 and FAS present equivalent performance.

The comparative results on selected MESH instances are summarised in Table 5.6. No statistical analysis is required in this case, as for each MESH instance, the worst results given by FAS are better than the best results given by H0-H4.

Table 5.5: Comparative Results with Individual Heuristics on Selected DIMACS and BHOSLIB Instances, the Best Success Rate Obtained for Each Instance is Highlighted in Bold.

(a)

Instance	H0			H1			H2		
	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)
frb56-25-2	**91**	552882515	772.36	0	n/a	n/a	0	n/a	n/a
frb59-26-1	**83**	844792024	1271.41	0	n/a	n/a	0	n/a	n/a
brock800_2	0	n/a	n/a	80	889349615	2125.75	0	n/a	n/a
Mann_a81	54	1939835273	4800.59	0	n/a	n/a	0	n/a	n/a

(b)

Instance	H3			H4			FAS		
	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)
frb56-25-2	0	n/a	n/a	0	n/a	n/a	**91**	741094488	969.44
frb59-26-1	0	n/a	n/a	0	n/a	n/a	77	1015432521	1478.78
brock800_2	**100**	22329070	53.3	**100**	43319358	104.01	**100**	133592306	239.12
Mann_a81	0	n/a	n/a	0	n/a	n/a	**97**	477085773	999.76

Table 5.6: Comparative Results with Individual Heuristics on Selected MESH Instances, the Best Average Solution and the Best Solution Found in Each Case are Highlighted in Bold.

(a)

Instance	n	H0			H1			H2		
		MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN
face	22871	12704	12701.9	12700	12663	12659.4	12659	12745	12735.6	12725
gameguy	42623	21915	21912.1	21910	21962	21951.2	21926	22057	22047.1	22034

(b)

Instance	n	H3			H4			FAS		
		MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN
face	22871	12766	12760.7	12752	12702	12698.6	12692	12658	**12656.1**	**12654**
gameguy	22032	22022.6	22007	22003	22073	22065.9	22063	21904	**21901.3**	**21899**

5.6.2.2 Comparison with Algorithm Using Random Heuristic Selection

To show the predictive measure as an effective approach for dynamically adapting heuristics, FAS is compared with the algorithm that selects the heuristic uniformly at random at the fixed interval. The algorithm using random heuristic selection operates on the identical set of heuristics as FAS which consists of H0 - H4. But unlike FAS which updates the new search heuristic according to the predictive measure, the algorithm using random heuristic selection selects the new search heuristic uniformly at random at each decision point.

The comparative results on selected instances from DIMACS and BHOSLIB benchmarks are summarised in Table 5.7(a). A statistical analysis has been performed to compare the significance of the results using the Wilcoxon rank-sum test [131], where p-value below 0.05 is considered to be statistically significant (confidence level 95%). Although FAS and the algorithm using random heuristic selection operate on the identical set of heuristics, it is clear that FAS achieves much better success rate on most instances, and is on par with the algorithm using random heuristic selection on brock800_2. In terms of the performance measured by run length in 100 independent executions on each instance, the statistical test shows that FAS significantly outperforms the algorithm using random heuristic selection.

Table 5.7: Comparative Results with the Algorithm Using Random Heuristic Selection on Selected DIMACS, BHOSLIB and MESH Instances.

(a) DIMACS and BHOSLIB, the Best Success Rate Obtained for Each Instance is Highlighted in Bold.

Instance	RANDOM			FAS		
	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)
frb56-25-2	20	1481457578	2079.2	**91**	741094488	969.44
frb59-26-1	10	14173757252	20997.9	77	1015432521	1478.78
brock800_2	100	726690275	139.7	100	133592306	239.12
Mann_a81	50	2630482830	2283.5	**97**	477085773	999.76

(b) MESH, the Best Average Solution and the Best Solution Found in Each Case are Highlighted in Bold.

Instance	RANDOM			FAS		
	MAX	AVG	MIN	MAX	AVG	MIN
face	12667	12662	12658	12658	**12656.1**	**12654**
gameguy	21914	21908	21902	21904	**21901.3**	**21899**

The comparative results on selected MESH instances are summarised in Table 5.7(b). A statistical analysis has been performed to compare the significance of the results using the Wilcoxon rank-sum test [131], where p-value below 0.05 is considered to be statistically significant (confidence level 95%). The test suggests that FAS is significantly superior to the algorithm using random heuristic selection on both face and gameguy, in terms of the best solutions found in 20 independent executions.

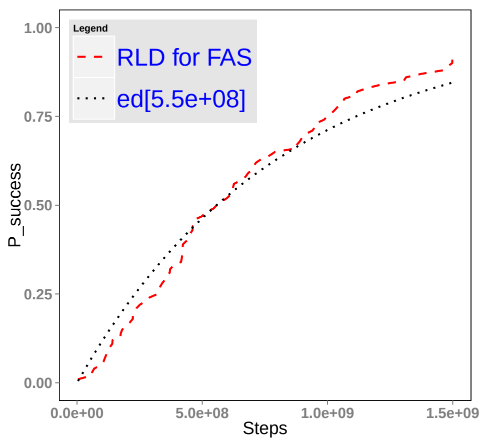
5.6.3 Characterising Run-time Behaviours of FAS

Due to the inherent randomness, the time needed by a randomised algorithm to find a solution differs from run to run even for a single instance. This variability of run time between multiple independent runs also demonstrates certain important characterisations for the behaviour of the algorithm. The run time of a randomised algorithm can be defined as a random variable which is fully characterised by the run-time distribution (RTD). Hoos et al. [62] proposed a RTD-based empirical analysis methodology for studying the run-time behaviour of algorithms, which has been widely applied to empirical analysis and performance modelling of heuristic algorithms.

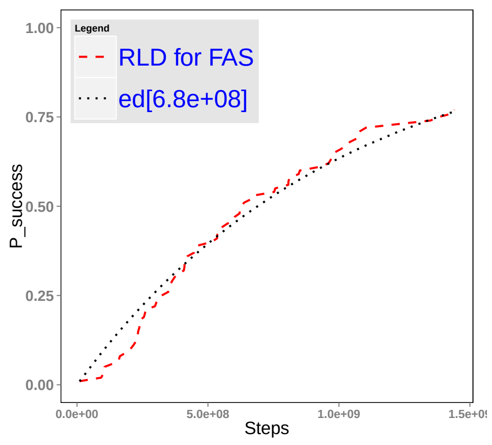
Here the RTD-based empirical analysis methodology has been applied for studying run-time behaviour of FAS on selected representative instances from different instance classes. For DIMACS benchmarks, $brock800\_2$ and MANN_a81 are selected, as they are the hardest instance from $brock$ and MANN family, respectively. For BHOSLIB benchmarks, $frb56$-25-2 and $frb59$-26-1 are selected, since they are of large size and challenging difficulty within the BHOSLIB benchmark.

The RTDs can be approximated using the cumulative form of an exponential distribution $ed[m](x) = 1 - 2^{-x/m}$, where m is the median of the distribution and x the number of steps required to find a solution. Note the representation ed[m] using $m = \ln 2/\lambda$ here is equivalent to the exponential distribution exp(λ), usually defined by $P(X \leq x) = 1 - e^{-\lambda x}$ in the statistical literature. Instead of measuring run-time distributions in terms of CPU-time, it is often preferable to use a more machine independent measure of an algorithm's performance such as representative operation counts. An appropriate operation count for FAS and other heuristic algorithms for MVC is the swap operation, commonly referred to as a step.

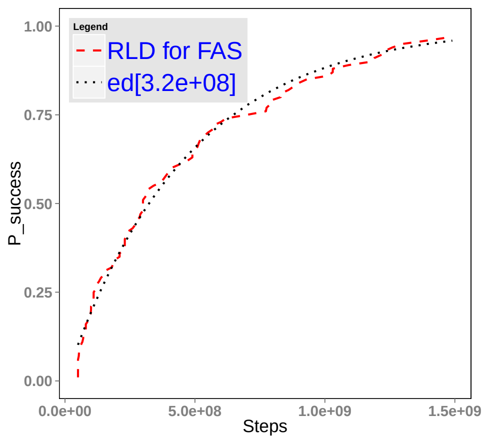
The RTDs for FAS on selected instances are illustrated in Figure 5.2 (based on 100 independent runs for each instance). From the diagrams it is clear to see that the RTDs are quite well approximated by exponential distributions. Similar exponential RTDs are observed for FAS applied to other DIMACS and BHOSLIB instances. For testing the goodness of these approximations the $Kolmogorov - Smirnov$ tests [85] have been used. The K-S test is a useful nonparametic test to compare a sample with a reference probability distribution. The K-S tests for the approximations fail to reject the null hypothesis that the sampled RTDs abide by the exponential distributions at a standard significance level $\alpha = 0.05$, with p-values at 0.8719 (frb56-25-2), 0.6643 (frb59-26-1), 0.7956 (MANN_a81) and 0.9671 ($brock$800_2).



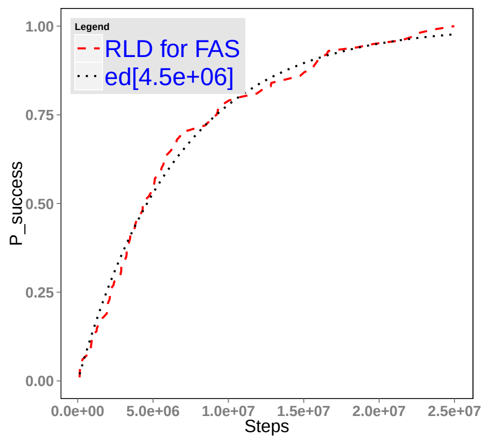
(a) frb56-25-2



(b) frb59-26-1



(c) MANN_a81



(d) brock800_2

Figure 5.2: RTDs of FAS Applied to the Selected Instances

Similar exponential RTDs have been observed for many other effective randomised algorithms, e.g., for Maximum Clique [97], for SAT [60, 61], for MAXSAT [108], and for scheduling [123]. By the arguments made in [62] for algorithms exhibiting an exponential RTD, the probability of finding a solution within a fixed time interval is independent of the run-time spent before, namely, the algorithms are essentially memoryless, as for a given total time t, restarting at time $t^{'} < t$ does not significantly influence the probability of finding a solution in time t. It is concluded, for FAS, that the probability of finding an optimal solution within a fixed amount of time (or steps) does not depend on the number of search steps done in the past. Consequently, they are robust w.r.t. the cut-off parameters like the maxsteps parameter and the number of random restarts.

5.6.4 Sensitivity Analysis of Adaptive Interval and Cut-off Parameter

This subsection performs additional empirical analysis for the adaptive interval parameter and the cut-off parameter, which are the only two parameters to tune before actually applying the algorithm.

5.6.4.1 Adaptive Interval

The adaptive interval determines when to change the dynamics of the algorithm. To study the impact of this parameter on the algorithm's performance, empirical comparison has been performed for four different adaptive interval values ranging from small to large: 5e+06, 1e+07, 2.5e+07, 5e+07 on selected instances from DIMACS, BHOSLIB and MESH. The experimental results are summarised in Table 5.8. The Friedman's test [40] is employed to determine if there are global difference in the results, where p-value below 0.05 is considered to be statistically significant (confidence level 95%). The results suggest that the differences between FAS with different values of adaptive interval are statistically insignificant. It is therefore concluded that the performance of FAS is robust with respect to the adaptive interval parameter.

Table 5.8: Performance of FAS in Different Adaptive Intervals on Selected DIMACS, BHOSLIB and MESH Instances

(a) Performance of FAS in Different Adaptive Intervals on Selected DIMACS and BHOSLIB Instances

Instance	5e+06			1e+07			2.5e+07			5e+07		
	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)	Suc	ERL	ERT(s)
frb56-25-5	90	587248497	810.7	91	741094488	969.44	88	966017282	1329.06	91	748535331	1024.28
frb59-26-1	80	1348983905	2007.8	77	1015432521	1478.78	70	1516226852	2202.7	68	1672182547	2439.3
brock800_2	90	341287890	586.67	100	133592306	239.12	93	341280769	563.5	100	101491996	193.8
Mann_a81	95	283767098	652.3	97	477085773	999.76	98	465275170	938.9	95	593001274	973.4

(b) Performance of FAS in Different Adaptive Intervals on Selected MESH Instances

Instance	n	5e+06			1e+07			2.5e+07			5e+07		
		MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN
face	22871	12661	12657.6	12655	12658	12656.1	12654	12658	12656.4	12654	12662	12658.8	12656
gameguy	42623	21909	21905.6	21902	21904	21901.3	21899	21911	21906.6	21907	21912	21907.4	21899

5.6.4.2 Cut-off Parameter

The cut-off parameter determines when to terminate the algorithm unless the optimal solutions are found before reaching the cut-off time. For DIMACS and BHOSLIB benchmarks where the optima are known, FAS terminates upon finding the optimal solution or reaching the cut-off condition which is set at 1.5e+09 steps. For MESH instances where the optima are unknown, FAS terminates upon reaching the cut-off condition which is set at 1.5e+09 steps.

Since RTDs of FAS exhibit an exponential distribution when applied to DIMACS and BHOSLIB benchmarks, the performance of FAS is robust w.r.t. the cut-off parameter and thus it is concluded that setting the cut-off parameter of FAS at 1.5e+09 steps is appropriate. To ensure fair comparison on MESH instances where the optimal solutions are unknown, an empirical analysis has been conducted to show the cut-off parameter is set appropriately. Figure 5.3 shows the solutions (Size of Independent Set) found for the largest instance of MESH as the number of steps increases for NuMVC, PLS and FAS, respectively. The three algorithms seem to converge after 1.5e+09 steps, which is set as the cut-off parameter.

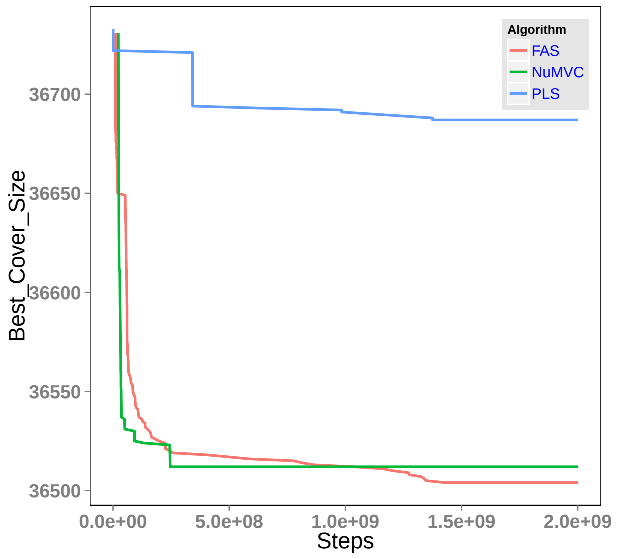


Figure 5.3: The Minimum Vertex Cover Found for the Largest Instance of MESH as the Number of Steps Increases for NuMVC, PLS and FAS.

5.6.5 Behaviours of FAS and the Predictive Measure

This subsection aims to perform additional empirical analysis to further investigate the behaviours of FAS and the predictive measure present on different instance classes.

To perform this empirical analysis, two representative instances are selected from each instance class. For DIMACS benchmarks, $brock$800_2 and MANN_a81 are selected, as they are the hardest instance from $brock$ and MANN family, respectively. For BHOSLIB benchmarks, frb56-25-2 and frb59-26-1 are selected, since they are of large size and challenging difficulty within the BHOSLIB benchmark. For MESH instances, $face$ and $gameguy$ are selected. $face$ is of reasonable size within MESH, whilst $gameguy$ is much larger and tends to be more difficult.

Pullan et al. [98] have noted that instances from DIMACS and BHOSLIB benchmarks have different characteristics. Furthermore, MESH instances are fundamentally different from DIMACS and BHOSLIB benchmarks [5]. Note the following conclusions on DIMACS are for the complementary DIMACS graphs.

- The DIMACS MANN instances have a large proportion of plateaus in the instance search space, and thus greedy heuristics are unsuitable to solve them.
- The DIMACS brock instances have minimum vertex covers that consist of medium to lower degree vertices, and are designed to defeat greedy heuristics.
- The BHOSLIB instances have minimum vertex covers consisting of vertices whose distribution of vertex degree closely matches that for the complete graph. These are difficult instances for both greedy and diversification heuristics
- The MESH instances are large, sparse graphs, mostly with average vertex degree three. The characteristics of the minimum vertex covers are unclear since MESH are arising from real-world problems where the optimal solutions are unknown.

5.6.5.1 On DIMACS

The results of the investigation on MANN_a81 and *brock*800_2 are illustrated in Figure 5.4(a) and Figure 5.4(b), respectively. In FAS, the best solution is updated if and only if the size of the new vertex cover obtained is smaller than or equal to the size of the current best solution.

The upper graph in Figure 5.4(a) illustrates the best solution found by different heuristics during the course of a FAS run on MANN_a81. As reported in Table 2, the existing state-of-the-art algorithms NuMVC and PLS only achieve success rate of 54% and 0% for MANN_a81, respectively. According to [98], this difficulty of MANN_a81 is due to the fact that MANN instances have a large proportion of plateaus in the search space, where little information can be obtained to guide the search for most heuristic algorithms. From the upper graph in Figure 5.4(a), it is clear to see that heuristics H1 and H2 of FAS are performing plateau moves on vertex covers of size $opt(MANN\_a81) - 2$ most of the time during the run, where $opt(MANN\_a81)$ denotes the optimal vertex cover size of MANN_a81. In FAS, the best solution found so far is taken as the starting point for the next heuristic being selected. The plateau moves performed by H1 and H2 has taken the search to the right region in the search space, which has has resulted in the convergence of FAS to the optimal solution.

With respect to behaviours of the predictive measure on MANN_a81, the lower graph in Figure 5.4(a) shows the probabilities for selecting different heuristics over the course of a FAS run on MANN_a81. These probabilities are normalised values of the predictive problem difficulty measure computed based on the latest performance of heuristics. On MANN_a81, H0 is the dominant heuristic with the probability for selecting it above 0.75 all the time, this is confirmed by the upper graph in Figure 5.4(a) that H0 quickly finds the vertex cover of size $opt(MANN\_a81) - 2$, and takes the search to escape from the plateau to converge to the optimum. In the meantime, despite the probabilities for selecting heuristics H1 and H2 are much lower than that of H0, the efficacy of H1 and H2 are essential for finding the minimum vertex cover of MANN_a81. As seen in the upper graph in Figure 5.4(a), the plateau moves performed by H0 and H1 take the search to more promising regions in the search space, as a consequence, enabling the escape from the plateau to convergence by H0.

The upper graph in Figure 5.4(b) illustrates the best solution found by different heuristics during the course of a FAS run on $brock800\_2$. It is noted in [98] that brock instances are deliberately designed to defeat greedy heuristics by explicitly incorpo-

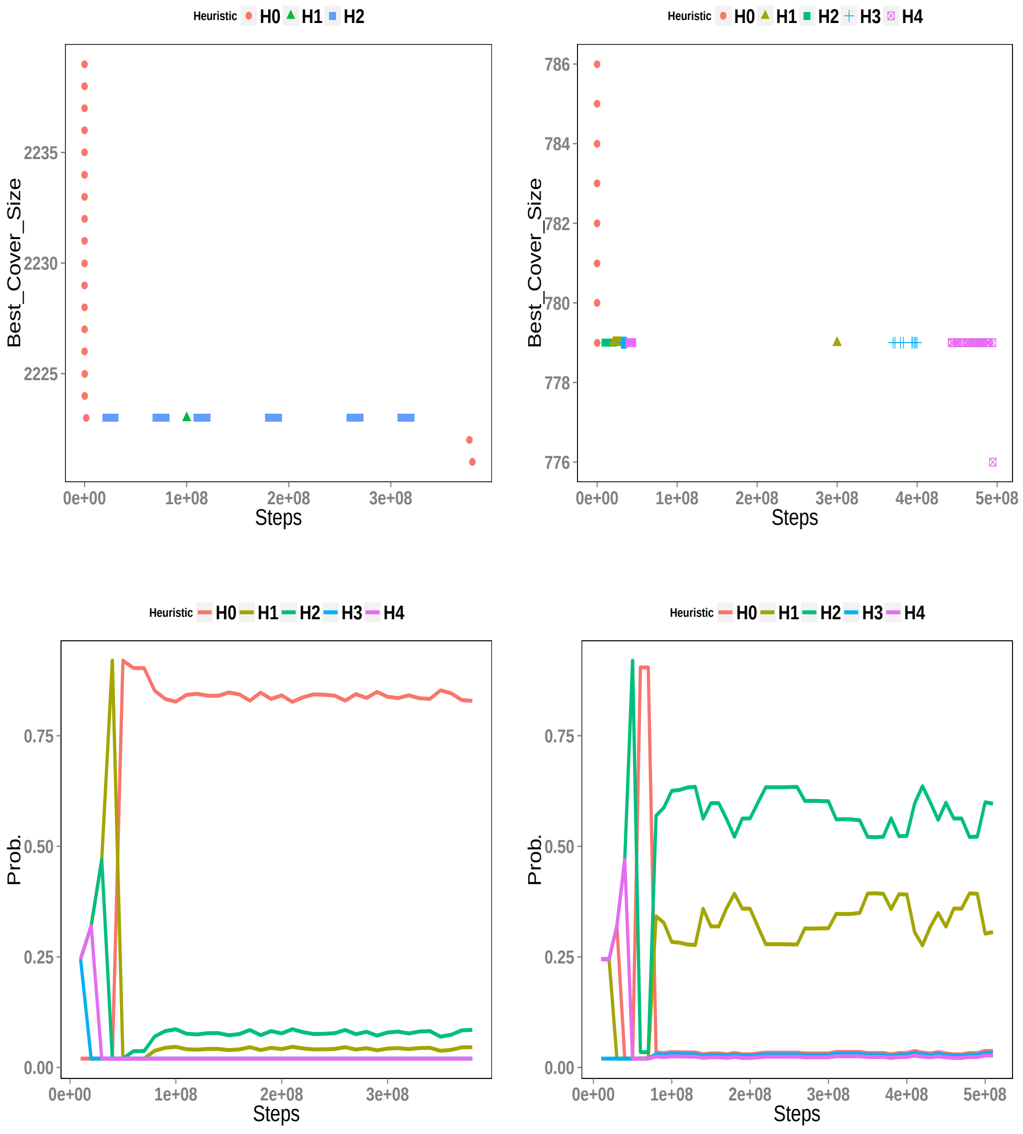

(a) MANN_a81 (b) brock800_2

Figure 5.4: Adaptive Behaviours of FAS and the Predictive Problem Difficulty Measure on the Selected DIMACS Instances.

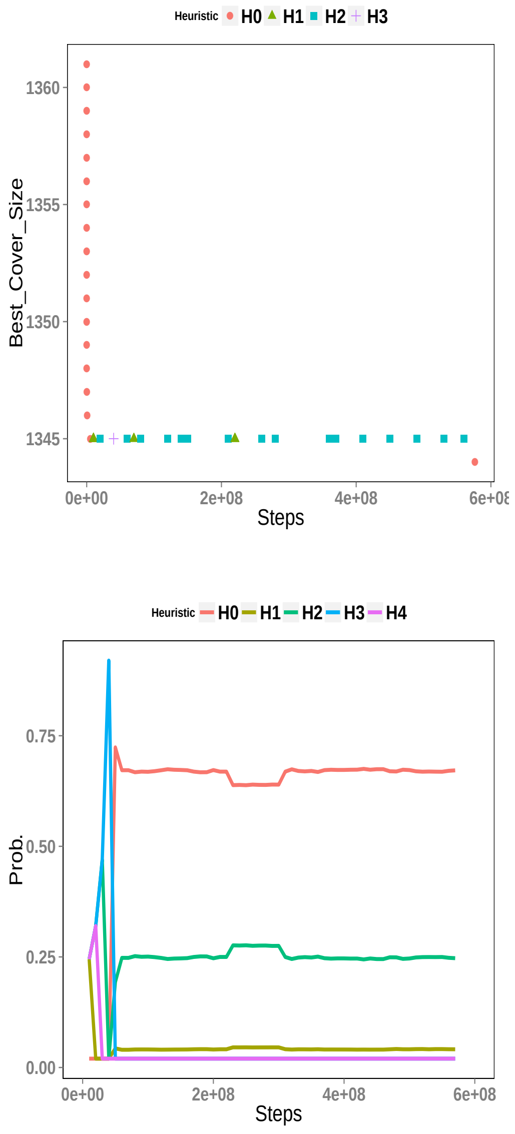
rating medium to lower degree vertices with the minimum vertex covers. Similarly to the observations on MANN_a81, in most of the time FAS is performing plateau moves on vertex cover of size $opt(brock800\_2) - 3$, where $opt(brock800\_2)$ denotes the optimal vertex cover size of $brock800\_2$. But unlike on MANN_a81, these plateau moves are performed by H1, H2, H3 and H4 and it is H4 that takes the search to escape from the plateau to converge to the optimum. It is worth noting that a large number of different vertex covers on the plateau are discovered, this indicates a large proportion of plateaus might exist in the search space of $brock800\_2$.

Regarding behaviours of the predictive measure on $brock800\_2$, the lower graph in Figure 5.4(b) shows the probabilities for selecting different heuristics over the course of a FAS run on $brock800\_2$. The pattern for the probabilities presents relatively large fluctuations, H1 and H2 have clear advantages, with probabilities over 0.25 and 0.5 most of the time, respectively. The probabilities for H3 and H4 are significantly smaller than that of H1 and H2. This pattern for the behaviours of the predictive measure is unique in a sense that it is very different from the behaviours of the predictive measure on other instances.

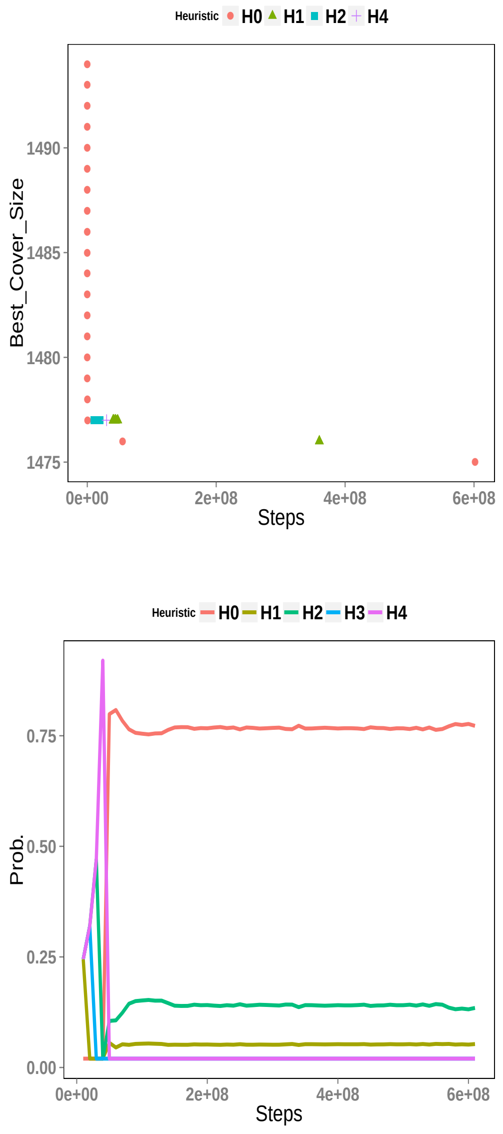
5.6.5.2 On BHOSLIB

The results of the investigation on frb56-25-2 and frb59-26-1 are illustrated in Figure 5.5(a) and Figure 5.5(b), respectively. The upper graphs in Figure 5.5(a) and Figure 5.5(b) illustrate the influence of different heuristics in terms of finding better/plateau solutions as FAS proceeds on frb56-25-2 and frb59-26-1, respectively. The patterns observed for behaviours of FAS on these two frb instances are very similar, which has confirmed that FAS recognises the two instances with very similar characteristics. H0 turns out to be the most critical heuristic for finding the optimal solutions of these two frb instances, this result is in line with our expectations since H0 is adopted from NuMVC which is the most effective algorithm in solving frb instances.

The lower graphs in Figure 5.5(a) and Figure 5.5(b) demonstrate the probabilities for selecting different heuristics over FAS runs on frb56-25-2 and frb59-26-1.



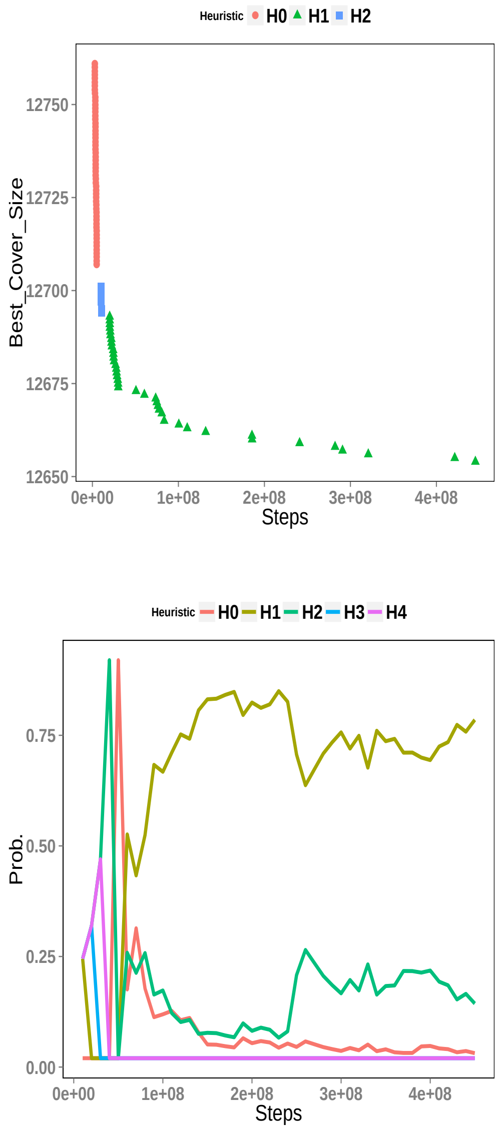
(a) frb56-25-2



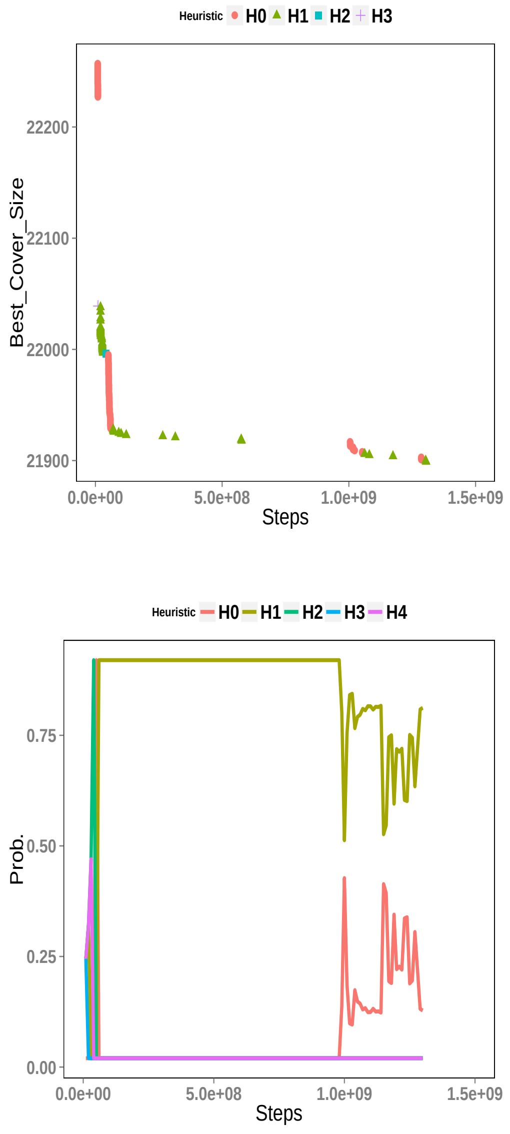
(b) frb59-26-1

Figure 5.5: Adaptive Behaviours of FAS and the Predictive Problem Difficulty Measure on the Selected BHOSLIB Instances.

Very similar patterns are observed again, H0 turns out to be the dominant heuristic with significantly higher probabilities to be selected than all other heuristics. In this case, FAS has identified H0 as the most effective heuristic for solving frb instances where H0 is known to be the best heuristic for frb instances so far. This has demonstrated that FAS is able to recognise the characteristics of the instance and match it with the most effective heuristic.



(a) face



(b) gameguy

Figure 5.6: Adaptive Behaviours of FAS and the Predictive Problem Difficulty Measure on the Selected MESH Instances.

5.6.5.3 On MESH

The results of the investigation on *face* and *gameguy* are illustrated in Figure 5.6(a) and Figure 5.6(b), respectively. MESH instances are fundamentally different from the previous benchmarks, which are large, sparse graphs with linear-sized vertex covers. The upper graphs in Figure 5.6(a) and Figure 5.6(b) show the influence of different heuristics in terms of finding better solutions on *face* and *gameguy*. It is noted that FAS starts from relatively bad solutions on both *face* and *gameguy*, which are then steadily improved by different heuristics iteratively, mainly H0, H1 and H2.

The lower graphs in Figure 5.6(a) and Figure 5.6(b) show the probabilities for selecting different heuristics of FAS on *face* and *gameguy*. The graphs present similar patterns which are different from the previous graphs. Here H1 has clear advantages, followed by H0 and H2. According to Tables 5.4 and 5.6, the state-of-the-art algorithms NuMVC and PLS only deliver very low-level solutions for MESH instances, and so do individual heuristics in FAS. The outstanding performance of FAS on MESH instances are the consequence of cooperation between different heuristics.

5.7 Summary

Adaptive heuristic algorithms have gained increasing popularity in recent years. This is motivated by the fact that there is evidence, both empirical and theoretical, that the most effective configuration of a given algorithm for solving a particular problem instance can vary during the search process. Also such algorithms with improved generality would increase the potential applicability for solving a broad class of problem instances with diverse characteristics. A common limitation in most existing adaptive heuristic algorithms lies in that these algorithms utilise historical information only while selecting from candidate configurations. However, an accurate selection must account for the fact that the optimal configuration could be regarded as a dynamic random variable and the underlying distribution of this random variable changes as the search proceeds.

To address this issue, a generic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure has been proposed. To demonstrate the effectiveness of the proposed approach, the proposed framework has been applied to design the fitness landscape based adaptive search algorithm (FAS) for tackling the minimum vertex cover problem (MVC). The experimental results showed that FAS is competitive with the state-of-the-art algorithms NuMVC and PLS on the widely studied DIMACS and BHOSLIB benchmarks. In particular, FAS significantly outperformed NuMVC and PLS on the putatively hard MANN_a81 instance, achieving a success rate of 97%. Furthermore, FAS achieved dominant performance over NuMVC and PLS on a new instance class MESH arising from real-world problems. Further empirical analysis on selected instances from DIMACS, BHOSLIB and MESH revealed that the predictive measure was essential to the adaptivity of FAS when solving a broad class of instances with diverse characteristics, where the candidate heuristics were prompted to cooperate for solving different instances accordingly.

Chapter 6

CONCLUSIONS

The overall goal of this thesis is to develop an effective approach for analysing fitness landscapes, which can then be used to build enhanced techniques for finding best algorithm configurations of metaheuristics for solving optimisation problems. The motivations are twofold. On one hand, finding the best suited configuration of a given algorithm for solving a particular problem requires in-depth understanding of the problem in relation to the algorithm, which can be provided by fitness landscape analysis. On the other hand, despite extensive research on fitness landscape analysis and a number of developed approaches, very few approaches are used in practice, due to the fact that the existing fitness landscape analysis approaches have either proved to be unreliable or are unable to be used in practice due to their own limitations (e.g. require known global optima).

The work presented in this thesis has contributed to fulfil this goal and addressed three important research issues in incorporating fitness landscape analysis to design an algorithm. In particular, a novel and effective approach for characterising fitness landscapes and measuring problem difficulty has been developed. Furthermore, the proposed approach, along with the existing fitness landscape analysis techniques, have been incorporated to build enhanced techniques for both static and dynamic algorithm configurations.

This chapter summarises the contributions presented in preceding chapters of the thesis with remarks for future work. Section 6.1 reiterates the need and importance to develop an effective and reliable approach for characterising fitness landscapes and measuring problem difficulty. The study presented in Chapter 3 is summarised which proposed the fitness-probability cloud as a novel approach for characterising fitness landscapes, and the accumulated escape probability for measuring problem difficulty, motivated by a theoretical investigation of the escape probability and the expected runtime. Section 6.2 summarises the motivation and conclusion of the study in chapter 4 that incorporated the problem difficulty measure, the accumulated escape probability, with classification algorithms to build automatic algorithm configuration methods on a per-instance base. Section 6.3 summarises the motivation and conclusion of the study in chapter 5 that incorporated the predictive problem difficulty measure to develop a generic framework for designing adaptive heuristic algorithms. Section 6.4 concludes this thesis with remarks for future work.

6.1 A Novel Approach for Characterising Fitness Landscapes and Measuring Problem Difficulty

Metaheuristics are extensively developed and applied in solving complex optimisation problems. Nevertheless, relatively little attention has been paid to study the implications behind the empirical results, i.e., on understanding which kinds of problems the proposed algorithm will perform well or poorly and why. Given this lack of understanding of problems in relation to algorithms, despite many successes of metaheuristics in solving complex optimisation problems, fundamental questions like which algorithms or algorithm variants are best suited for solving a given problem remain unanswered. Recent theoretical investigations such as runtime analysis [68] have made some progress in addressing this problem, where on particular problem instances the EA-hardness [58], when a large population is useful [56] and the interactions between mutation and selection [77] are investigated. However, theoretical investigations are often under several assumptions which cannot be met in practical scenarios, finding the best suited algorithm for solving a particular problem remains a daunting task for practitioners.

Fitness landscape analysis, a powerful analytical tool particularly in understanding characteristics of optimisation problems and the associated behaviours of metaheuristics, shows great promise in overcoming this long standing challenge. Although extensive studies have been conducted on fitness landscape analysis to develop generally effective techniques, the developed techniques have either proved to be unreliable or are unable to be used in practice due to their own limitations (e.g. require the global optima to be known), and there is a lack of an effective and reliable approach for characterising fitness landscapes and measuring problem difficulty.

This thesis fulfilled this need through proposing a fitness-probability cloud as a novel approach for characterising fitness landscapes, and the accumulated escape probability to explicitly quantify the problem difficulty with respect to algorithms, based on the notion of escape probability arising from time complexity studies of metaheuristics.

In particular, this thesis has for the first time bridged the gap between time complexity studies of metaheuristics and fitness landscape analysis, where the notion of escape probability has been formally defined and a theoretical investigation has been conducted to show that the escape probability is a critical factor in determining lower bounds of the expected runtime, which is usually taken as a difficulty measure in time complexity studies of metaheuristics. Two further developments have been obtained based on the escape probability. First, the fitness-probability cloud [82] has been defined to obtain an overall characterisation of fitness landscapes. Second, the accumulated escape probability (*aep*) [82] has been derived from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms.

Results from the case studies showed that *aep* outperformed the negative slope coefficient, which is a widely applied problem difficulty measure in GP and GA, and can serve as a reliable predictive problem difficulty measure for discriminating the relative problem difficulty of the subset sum problem, onemax, onemix and trap w.r.t. the mutation-based (μ+λ) EAs. The proposed fitness-probability cloud and *aep* measure showed great promise in effectively characterising fitness landscapes and reliably measuring problem difficulty. Since the fitness-probability cloud and *aep* are generic approaches which do not require any a priori knowledge, and the definition of *aep* is general and has many degrees of freedom, the fitness-probability cloud and *aep* can serve as an effective and reliable approach for fitness landscape analysis, and can potentially be incorporated to build enhanced techniques for finding the best suited algorithm for solving a particular problem.

6.2 Incorporating Fitness Landscape Analysis for Static Algorithm Configuration

It is widely acknowledged that finding good algorithm configurations is essential to obtain robust and high algorithm performance, also it has been observed that a given algorithm requires different configurations in order to find good solutions for different problem instances [41, 94, 137]. The static algorithm configuration concerns determining a priori the most effective configuration of a given algorithm for solving a particular problem instance.

Commonly the static algorithm configuration problem is to find the best suited configuration in a finite set of candidate configurations, which is traditionally formulated as an optimisation problem. But unlike standard optimisation problems, the objective function of the static algorithm configuration problem cannot be written analytically and is typically highly non-linear and very expensive to compute.

The most common technique for selecting the best suited configuration is by trial and error, which is not only inefficient and laborious, but very unlikely to locate the optimal configuration due to the size of the configuration space. Recent years have seen many approaches proposed to automatically determine a priori the best suited configuration of a given algorithm for solving a particular problem. In particular, most previous approaches are either generic (problem-independent) but only produce a one-size-fits-all configuration for an entire set of problem instances, or perform per-instance configuration only for a particular problem by making use of problem-specific features. To the best of our knowledge, there exists very few generic (problem-independent) techniques for automatically configuring a given algorithm on a per-instance basis.

This thesis addressed this need in two steps. First, the static algorithm configuration problem has been reformulated as a decision problem, which is to determine whether a given algorithm configuration is suitable for solving a given problem instance. This reformulation is in the interest of practice since it is usually infeasible to explore the entire configuration space for finding the optimal configuration, it is somewhat sufficient if an approximation solution can be efficiently identified within a finite set of configurations. Second, a problem difficulty measure, the accumulated escape probability, has been incorporated to build a generic approach which performs automatic algorithm configuration on a per-instance base [78, 81]. The proposed approach is based on learning the pattern which governs the relationship between the configurations of a given algorithm and the characteristics of problem instances, where the accumulated escape probability has been employed as features to characterise problem instances.

Results from the case study on the Unique Input Output Sequence problem (UIO) showed the proposed approach can reliably determine whether a configuration of the $(\mu + \lambda)$ EA is suitable for solving a UIO instance, provided a configuration was considered as effective when the $(\mu + \lambda)$ EA executed under the configuration showed high performance. Furthermore, within a finite set of configurations, the proposed approach significantly outperformed ParamILS, which is a state-of-the-art automatic algorithm configuration method, on automatically configuring the $(\mu + \lambda)$ EAs for solving the UIO problem.

Further to finding the best suited algorithm configuration within a finite set of existing algorithm configurations, it is always useful to produce a novel, problem-specific configuration (e.g. new heuristic/search operator) which potentially outperforms other configurations on a particular class of problem instances. This thesis has established a bridge connecting results of theoretical fitness landscape analysis and design of novel, effective heuristics, through proposing an approach to perform elementary landscape analysis and further explicitly apply the analytical results to develop efficient local search heuristics [80]. Results from the case study illustrated that the proposed elementary hill climbing algorithm developed using results from elementary landscape analysis significantly outperformed both the standard stochastic hill climber and the state-of-the-art in solving the next release problem in software engineering.

6.3 Incorporating Fitness Landscape Analysis for Dynamic Algorithm Configuration

Traditionally algorithm configuration methods attempt to determine a priori the most appropriate configuration of a given heuristic algorithm for solving a particular problem instance. However, there are both empirical and theoretical evidence showing that the most effective configuration of a given algorithm for solving a particular problem instance can vary during the search process [116]. This motivates the development of heuristic algorithms that dynamically adapt their configurations (search operators, numerical parameters, etc.) during the search process. These algorithms, often referred to as adaptive heuristic algorithms, aim to identify and select the most effective configuration of a given algorithm at each decision point during the search. In particular, the behaviours of such heuristic algorithms are adapted to specific characteristics of problem instances, and the generality of such algorithms would be improved to be able to solve a broad class of problems with diverse characteristics. A common limitation in most existing adaptive heuristic algorithms lies in that these algorithms utilise historical information only while selecting from candidate configurations. It is noted that considering only the past performance can be misleading, as the optimal configuration could be regarded as a dynamic random variable and the underlying distribution of this random variable changes as the search proceeds.

This thesis addressed this vital issue through incorporating predictive information provided by the predictive problem difficulty measure to design a systematic mechanism for adaptive heuristic search. A generic framework for designing adaptive heuristic algorithms using the predictive problem difficulty measure has been proposed, in which the predictive measure can predict the expected performance of candidate configurations by exploiting information extracted from the search trajectory, therefore selection of the best suited configuration can be based on the expected performance instead of past performance only. As a result, a more accurate mapping between candidate configurations of a given algorithm and specific characteristics of search space can be established.

Results from the case study, which applied the proposed framework to develop the fitness landscape based adaptive search algorithm (FAS) for the minimum vertex cover problem (MVC), showed that FAS either outperformed or was comparable to the state-of-the-art algorithms for MVC on both widely studied benchmarks and real-world instances. New lower bounds have been found by our algorithm on several hard problem instances. Further empirical analysis revealed that the predictive measure was essential to the adaptivity of FAS when solving a broad class of instances with diverse characteristics, where the candidate heuristics were prompted to cooperate for solving different instances accordingly.

6.4 Future Work

The future research identified for this thesis proceed along several directions.

6.4.1 Exploiting Fitness-Probability Cloud

The fitness-probability cloud provides an overall characterisation of fitness landscapes, where the critical information on the underlying fitness-probability cloud can be exploited to further understand the problem difficulty with respect to the algorithm applied to solve it. The accumulated escape probability can be regarded as the first attempt to extract information from the fitness-probability cloud to explicitly quantify the problem difficulty with respect to algorithms. However, the definition of the accumulated escape probability is simple and intuitive, there is large room for research efforts in attempting to extract the information on the fitness-probability cloud in many alternative ways, e.g., by considering the importance of different fitness values instead of treating them equally.

6.4.2 Identifying Effective Application Domains for Problem Difficulty Measures

Assuming a worst-case perspective, He et al. [54] have rigorously proved for both approximate and exact measures, the predictive versions that can be computed in polynomial-time do not exist unless P = NP or BPP = NP. This result has concluded that finding a useful predictive problem difficulty measure in general is impossible. In this thesis, the accumulated escape probability derived from the fitness-probability cloud has been proposed as a predictive problem difficulty measure, although the results from case studies on unitation functions and the subset sum problem suggested it is a reliable measure, if we refer to the theoretical conclusion [54] it is obvious to see that the accumulated escape probability cannot be useful in all problem domains. For the sake of practitioners to apply this measure, an important issue to address in the future work of this thesis would be to identify the problem domains where the proposed problem difficulty measure, accumulated escape probability, is reliably useful, and to further determine the limitations of the predictive problem difficulty measure which can potentially lead to implications on how to improve and design an enhanced problem difficulty measure.

6.4.3 Exploring Theoretical Results for Fitness Landscape Analysis

Fitness landscape analysis aims to gain in-depth understanding of problem characteristics and thus determining the problem difficulty with respect to algorithms, which is fundamentally linked with theoretical investigations in the field of metaheuristics, such as time complexity studies of metaheuristics. Fitness-probability cloud served as the first attempt to bridge theoretical results in the field of metaheuristics and fitness landscape analysis. In fact, there exists a number of theoretical studies such as convergence [46] and runtime analysis [68] on understanding the relationship between problem characteristics and behaviours of search algorithms. It is very promising to investigate a wider spectrum of theoretical studies and find out whether the results can be applied to build enhanced approaches for fitness landscape analysis or not.

6.4.4 Extending the Proposed Automatic Algorithm Configuration Method

The automatic algorithm configuration method proposed in Chapter 4 appeared to be the first generic approach to automatically configure a heuristic algorithm on a per-instance base. This approach was studied on configuring EAs for solving the UIO problem in software testing, since the features (a set of problem difficulty measures) used to represent the instances are problem-independent, it is very interesting to apply the proposed approach to a wider class of problems, particularly software engineering problems. Search based software engineering [53] is emerging as a popular approach which applies metaheuristics to tackle software engineering problems, but despite a large number of algorithms being developed, it is difficult for software engineers to apply these algorithms since finding good configurations of a given algorithm requires extensive knowledge about the algorithm. The proposed automatic algorithm configuration method is a promising solution to address this issue.

Furthermore, the proposed automatic algorithm configuration is based on learning the pattern which governs the relationship between the configurations of the algorithm and the characteristics of instances. The choice of the classification algorithm is essential to the performance of the algorithm configuration method, we start with the support vector machine, but more classification algorithms can be tested and compared. In addition, automatic algorithm configuration which takes into account the characteristics of the problem instance and past performance on similar instances lends itself nicely to incremental learning. In an incremental setting, it is very interesting to study the trade off between exploitation (choosing the best suited configuration) and exploration (choosing other configurations from which the classifier can learn more about poorly known configuration regimes).

6.4.5 Extending the Proposed Framework for Designing Adaptive Heuristic Algorithms

Although effective adaptive heuristic algorithms require problem-specific algorithm configurations (e.g. search operators/heuristics), the proposed framework does introduce a generic measure to design a systematic mechanism for adaptive heuristic search. This generic framework, along with problem-specific algorithm configurations, can be applied to a wider spectrum of combinatorial optimisation problems.

In terms of improving the proposed framework, the previous work focussed on evaluating the performance of candidate heuristics using the problem difficulty measure, however, the selection mechanism is also important to the success of an adaptive heuristic algorithm. It is necessary to carry on investigating the influence of the selection mechanism on the algorithm performance by comparing different selection mechanisms.

References

[1] B. Adenso-Diaz and M. Laguna. Fine-Tuning of Algorithms Using Fractional Experimental Design and Local Search. *Operations Research*, **54**[1]:99–114, 2006. 3, 32, 36, 73

[2] A. V. Aho, A. T. Dahbura, D. Lee, and M. Uyar. An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours. In Richard Jerry Linn and M. Ümit Uyar, editors, *Conformance testing methodologies and architectures for OSI protocols*, pages 427–438. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995. 84

[3] L. Altenberg. Fitness distance correlation analysis: An instructive counterexample. In *In Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 57–64. Morgan Kaufmann, 1997. 23, 45, 49

[4] C. Ambroise and G. J. McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences*, **99**[10]:6562–6566, May 2002. 89

[5] D. Andrade, M.C. Resende, and R. Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, **18**[4]:525–547, 2012. 126, 131, 146

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, **47**[2-3]:235–256, May 2002. 44

[7] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, **2**, pages 1777–1784 Vol. 2, 2005. 132

[8] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and Software Technology*, **43**[14]:883 – 890, 2001. 74, 99, 100, 105

[9] J. W. Barnes, B. Dimova, S. P. Dokov, and A. Solomon. The theory of elementary landscapes. *Applied Mathematical Letters*, **16**:337–343, 2002. 27

[10] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, **29**[4]:610–637, 2001. 122

[11] M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004. 32, 73

[12] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. 3, 36

[13] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The Maximum Clique Problem. In *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, 1999. 122

[14] Y. Borenstein and R. Poli. Fitness distributions and ga hardness. In *Parallel Problem Solving from Nature - PPSN VIII*, **3242** of *Lecture Notes in Computer Science*, pages 11–20. Springer Berlin Heidelberg, 2004. 4, 20

[15] Y. Borenstein and R. Poli. Information Landscapes and Problem Hardness. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1425–1431, New York, NY, USA, 2005. ACM. 4, 20

[16] Y. Borenstein and R. Poli. Kolmogorov complexity, Optimization and Hardness. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 112–119, 2006. 25

[17] S. Cai, K. Su, and Q. Chen. Ewls: A new local search for minimum vertex cover. In *Proceedings of the national conference on Artificial intelligence*, AAAI'10, 2010. 122

[18] S. Cai, K. Su, and A. Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artif. Intell.*, **175**[9-10]:1672–1696, 2011. 122

[19] S. Cai, K. Su, and A. Sattar. Two new local search strategies for minimum vertex cover. In *Proceedings of the national conference on Artificial intelligence*, AAAI'12, 2012. 122, 124, 134, 139

[20] S. Cathabard, P. K. Lehre, and X. Yao. Non-uniform Mutation Rates for Problems with Unknown Solution Lengths. In *Foundations of Genetic Algorithms XI*, 2011. In Press. 87

[21] J. Cavazos and M. F. P. O'Boyle. Method-specific dynamic compilation using logistic regression. *SIGPLAN Not.*, **41**[10]:229–240, October 2006. 37

[22] F. Chicano, L. D. Whitley, and E. Alba. A methodology to find the elementary landscape decomposition of combinatorial optimization problems. *Evol. Comput.*, **19**[4]:597–637, December 2011. 29, 30, 45

[23] B. Codenotti and L. Margara. Local properties of some np-complete problems. Technical report, ICST, 1992. 27

[24] D. Corne, M. Oates, and D. Kell. On Fitness Distributions and Expected Fitness Gain of Mutation Rates in Parallel Evolutionary Algorithms. In *Parallel Problem Solving from Nature — PPSN VII*, **2439** of *Lecture Notes in Computer Science*, pages 132–141. Springer Berlin Heidelberg, 2002. 3, 43, 113

[25] C. Cortes and V. Vapnik. Support-Vector Networks. *Mach. Learn.*, **20**[3]:273–297, September 1995. 82

[26] J. C. Culberson. On the futility of blind search: An algorithmic view of no free lunch. *Evol. Comput.*, **6**[2]:109–127, June 1998. 2

[27] L. DaCosta, A. Fialho, M. Schoenauer, and M. Sebag. Adaptive Operator Selection with Dynamic Multi-armed Bandits. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 913–920, New York, NY, USA, 2008. ACM. 44, 113

[28] Y. Davidor. Epistasis Variance: A Viewpoint on GA-hardness. In *Rawlins, G. J. E., editor, Foundations of Genetic Algorithms (FOGA)*, pages 23–35, 1991. 4, 21, 120

[29] L. D. Davis and M. Mitchell. Handbook of Genetic Algorithms. *Van Nostrand Reinhold*, 1991. 3, 38, 42, 43, 113

[30] K. Deb and D. E. Goldberg. Analyzing Deception in Trap Functions. In *Foundations of Genetic Algorithms, 2*, pages 93–108, 1993. 4, 18, 43, 62, 113

[31] K. DeJong. *Parameter Setting in EAs: a 30 Year Perspective*, pages 1–18. Springer Verlag, 2007. 41

[32] K. Derderian, R. M. Hierons, M. Harman, and Q. Guo. Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs. *The Computer Journal*, **49**, 2006. 85

[33] S. Ding and B. Qi. Research of granular support vector machine. *Artificial Intelligence Review*, **38**[1]:1–7, 2012. 82

[34] O. J. Dunn. Multiple Comparisons Among Means. *Journal of the American Statistical Association*, **56**[293]:pp. 52–64, 1961. 107

[35] Á. Fialho, M. Schoenauer, and M. Sebag. Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 779–786, New York, NY, USA, 2009. ACM. 39, 41, 43

[36] A. Finkelstein, M. Harman, S. A. Mansouri, J. Ren, and Y. Zhang. A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. *Requir. Eng.*, **14**[4]:231–245, 2009. 105

[37] S. Forrest and M. Mitchell. Relative Building-Block Fitness and the Building-Block Hypothesis. In Darrell L. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126, San Mateo, CA, 1993. Morgan Kaufmann. 102

[38] S. Forrest and M. Mitchell. What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation. *Mach. Learn.*, **13**:285–319, November 1993. 4, 18

[39] J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: local search topology. *J. Artif. Int. Res.*, **7**[1]:249–281, 1997. 28

[40] M. Friedman. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, **32**[200]:675–701, 1937. 107, 135, 144

[41] M. Gagliolo and J. Schmidhuber. Dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, **47**:3–4, 2006. 7, 32, 72, 158

[42] S. Garcia, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, **15**[6]:617–644, 2009. 107

[43] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 1

[44] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition, January 1989. 4, 18

[45] D. E. Goldberg. Probability matching, the magnitude of reinforcement, and classifier system bidding. *Mach. Learn.*, **5**[4]:407–425, October 1990. 3, 43

[46] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. San Francisco, CA: Morgan Kaufmann, 1991. 49, 163

[47] J. Gratch and G. DeJong. COMPOSER: a probabilistic solution to the utility problem in speed-up learning. In *Proceedings of the tenth national conference on Artificial intelligence*, AAAI'92, pages 235–240. AAAI Press, 1992. 35

[48] A. Grosso, M. Locatelli, and W. Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, **14**[6]:587–612, December 2008. 122

[49] L. K. Grover. Local search and the local structure of np-complete problems. *Oper. Res. Lett.*, **12**[4]:235–243, October 1992. 25, 27, 96

[50] Q. Guo, R. M. Hierons, M. Harman, and K. Derderian. Computing Unique Input/Output Sequences Using Genetic Algorithms. In *In Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software, volume 2931 of LNCS*, pages 169–184. Springer, 2004. 85, 86

[51] J. Hallam and A. Prugel-Bennett. Large Barrier Trees for Studying Search. *Evolutionary Computation, IEEE Transactions on*, **9**[4]:385–397, 2005. 18, 20, 21, 49

[52] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. 89

[53] M. Harman and B. F. Jones. Search-Based Software Engineering. *Information and Software Technology*, **43**:833–839, 2001. 85, 164

[54] J. He, C. Reeves, C. Witt, and X. Yao. A Note on Problem Difficulty Measures in Black-box Optimization: Classification, Realizations and Predictability. *Evol. Comput.*, **15**[4]:435–443, December 2007. 23, 45, 49, 51, 114, 162, 163

[55] J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, **127**[1]:57 – 85, 2001. 115

[56] J. He and X. Yao. From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, **6**[5]:495–511, 2002. 2, 156

[57] J. HE AND X. YAO. Towards an Analytic Framework for Analysing the Computation Time of Evolutionary Algorithms. *Artificial Intelligence*, **145**:59–97, 2003. 115

[58] J. HE AND X. YAO. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, **3**[1]:21–35, 2004. 2, 156

[59] T. HONG, H. WANG, AND W. CHEN. Simultaneously Applying Multiple Mutation Operators in Genetic Algorithms. *Journal of Heuristics*, **6**:439–455, 2000. 3, 43, 113

[60] H. HOOS AND T. STÜTZLE. Local search algorithms for sat: An empirical evaluation. *J. Autom. Reason.*, **24**[4]:421–481, May 2000. 143

[61] H. HOOS AND T. STÜTZLE. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. 143

[62] H. H. HOOS AND T. STÜTZLE. Towards a characterisation of the behaviour of stochastic local search algorithms for sat. *Artificial Intelligence*, **112**[1–2]:213 – 232, 1999. 141, 143

[63] J. HORN AND D. E. GOLDBERG. Genetic Algorithm Difficulty and the Modality of Fitness Landscapes. In *Foundations of Genetic Algorithms, 3*, pages 243–269. Morgan Kaufmann, 1995. 4, 18, 19, 45, 49

[64] J. HORN, D. E. GOLDBERG, AND K. DEB. Long path problems. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, pages 149–158, London, UK, UK, 1994. Springer-Verlag. 19, 45, 49

[65] F. Hutter and Y. Hamadi. Parameter Adjustment Based on Performance Prediction: Towards an Instance-Aware Problem Solver. Technical report, Microsoft Research, Cambridge, UK, December 2005. 37

[66] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *J. Artif. Int. Res.*, **36**[1]:267–306, September 2009. 3, 5, 8, 9, 15, 32, 33, 36, 46, 76, 89, 91

[67] T. Jansen. On classifications of fitness functions. In *Theoretical aspects of evolutionary computing*, pages 371–385, London, UK, UK, 2001. Springer-Verlag. 22, 23, 45, 49

[68] T. Jansen and I. Wegener. On the Analysis of Evolutionary Algorithms — A Proof That Crossover Really Can Help. In Jaroslav Nešetřil, editor, *Algorithms - ESA' 99*, **1643** of *Lecture Notes in Computer Science*, pages 184–193. Springer Berlin Heidelberg, 1999. 2, 49, 156, 163

[69] D. J. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. American Mathematical Society, Boston, MA, USA, 1996. 122, 126

[70] T. Jones and S. Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 4, 22, 119

[71] B. A. Julstrom. What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 81–87, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 3, 42

[72] L. Kallel, B. Naudts, and M. Schoenauer. On functions with a fixed fitness-distance relation. In *Proceedings of 1999 Congress on Evolutionary Computation*, page 1910–1916, Piscataway, NJ, USA, 1998. IEEE Press. 23, 49

[73] R. Karp. Reducibility Among Combinatorial Problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 219–241. Springer Berlin Heidelberg, 2010. 121

[74] D. Lee and M. Yannakakis. Testing Finite-State Machines: State Identification and Verification. *IEEE Transactions on computers*, **43**[3]:30–320, 1994. 74, 84

[75] P. K. Lehre and X. Yao. Runtime Analysis of (1+1) EA on Computing Unique Input Output Sequences. In *IEEE Congress on Evolutionary Computation, 2007*, pages 1882 –1889, 2007. 85, 86

[76] P. K. Lehre and X. Yao. Crossover Can Be Constructive When Computing Unique Input Output Sequences. In *Simulated Evolution and Learning*, **5361** of *Lecture Notes in Computer Science*, pages 595–604. Springer, 2008. 85, 86

[77] P. K. Lehre and X. Yao. On the impact of the mutation-selection balance on the runtime of evolutionary algorithms. In *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, FOGA '09, pages 47–58, New York, NY, USA, 2009. ACM. 2, 157

[78] J. Li, G. Lu, and X. Yao. Fitness Landscape-based Parameter Tuning Method for Evolutionary Algorithms for Computing Unique Input Output Sequences. In *Proceedings of the 18th international conference on Neural In-*

formation Processing - Volume Part II, ICONIP'11, pages 453–460, Berlin, Heidelberg, 2011. Springer-Verlag. viii, 8, 92, 110, 159

[79] Lindawati, H. Lau, and D. Lo. Instance-based parameter tuning via search trajectory similarity clustering. In *Proceedings of the 5th international conference on Learning and Intelligent Optimization*, LION'05, pages 131–145, Berlin, Heidelberg, 2011. Springer-Verlag. 32, 35, 73

[80] G. Lu, R. Bahsoon, and X. Yao. Applying elementary landscape analysis to search-based software engineering. In *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, SSBSE '10, pages 3–8, Washington, DC, USA, 2010. IEEE Computer Society. 9, 111, 160

[81] G. Lu, J. Li, , and X. Yao. Fitness Landscape and Problem Difficulty in Evolutionary Algorithms: From Theory to Applications. In Andries Engelbrecht Hendrik Richter, editor, *Recent advances in the theory and application of fitness landscapes*. Springer-Verlag, 2013. viii, 8, 53, 92, 110, 159

[82] G. Lu, J. Li, and X. Yao. Fitness-probability Cloud and a Measure of Problem Hardness for Evolutionary Algorithms. In *Proceedings of the 11th European conference on Evolutionary computation in combinatorial optimization*, EvoCOP'11, pages 108–117, Berlin, Heidelberg, 2011. Springer-Verlag. vi, viii, 7, 20, 56, 57, 58, 59, 65, 66, 67, 78, 115, 116, 157

[83] N. Madras. *Lectures on Monte Carlo Methods*. American Mathemataical Society, January 2002. 60

[84] B. Manderick, M. K. Weger, and P. Spiessens. The Genetic Algorithm and the Structure of the Fitness Landscape. In *ICGA'91*, pages 143–150, 1991. 4, 119

[85] Jr. F. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, **46**[253]:pp. 68–78, 1951. 142

[86] J. Maturana, F. Lardeux, and F. Saubion. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, **16**[6]:881–909, December 2010. 14

[87] J. Maturana and F. Saubion. A compass to guide genetic algorithms. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, pages 256–265, Berlin, Heidelberg, 2008. Springer-Verlag. 42

[88] O. J. Mengshoel, D. E. Goldberg, and D. C. Wilkins. Deceptive and Other Functions of Unitation as Bayesian Networks. In *Symposium on Genetic Algorithms (SGA)*, 1998. 43, 61, 62, 113

[89] S. Minton. An analytic learning system for specializing heuristics. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 2*, IJCAI'93, pages 922–928, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. 3, 36

[90] H. Mühlenbein. How Genetic Algorithms Really Work: Mutation and Hill-climbing. In *PPSN*, pages 15–26, 1992. 38, 43, 112, 113

[91] B. Naudts and L. Kallel. A Comparison of Predictive Measures of Problem Difficulty in Evolutionary Algorithms. *Evolutionary Computation, IEEE Transactions on*, **4**[1]:1 –15, April 2000. 18, 45, 49

[92] P. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, **4**[3]:281–293, 2007. 51

[93] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, Mineola, NY., 1998. 50

[94] D. J. Patterson and H. Kautz. Auto-walksat: A self-tuning implementation of walksat. *Electronic Notes in Discrete Mathematics*, **9**[0]:360 – 368, 2001. 7, 32, 37, 72, 158

[95] R. Poli and L. Vanneschi. Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1335–1342, New York, NY, USA, 2007. ACM. 18

[96] W. Pullan. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization*, **6**[2]:214 – 219, 2009. 134

[97] W. Pullan and H. Hoos. Dynamic local search for the maximum clique problem. *J. Artif. Int. Res.*, **25**[1]:159–185, February 2006. 124, 143

[98] W. Pullan, F. Mascia, and M. Brunato. Cooperating local search for the maximum clique problem. *Journal of Heuristics*, **17**[2]:181–199, April 2011. 146, 147, 148

[99] W. J. Pullan. Phased local search for the maximum clique problem. *J. Comb. Optim.*, **12**[3]:303–323, 2006. 122, 134

[100] R.J. Quick, V.J. Rayward-Smith, and G.D. Smith. Fitness distance correlation and ridge functions. In AgostonE. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature — PPSN V*, **1498** of *Lecture Notes in Computer Science*, pages 77–86. Springer Berlin Heidelberg, 1998. 23, 45, 49

[101] N. J. Radcliffe and P. D. Surry. Fitness Variance of Formae and Performance Prediction. In *Whitley, L. D. and Vose, M. D., editors, Foundations of Genetic Algorithms (FOGA) 3*, pages 51–72, 1995. 4, 119

[102] C. R. Reeves and C. Wright. Epistasis in genetic algorithms: An experimental design perspective. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 217–224, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 23, 45, 49

[103] C.R. Reeves. Predictive measures for problem difficulty. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, **1**, page 743, 1999. 23

[104] S. Richter, M. Helmert, and C. Gretton. A stochastic local search approach to vertex cover. In *Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*, KI '07, pages 412–426, Berlin, Heidelberg, 2007. Springer-Verlag. 122

[105] S. Rochet, G. Venturini, M. Slimane, and E.M. Kharoubi. A critical and empirical study of epistasis measures for predicting ga performances: A summary. In Jin-Kao Hao, Evelyne Lutton, Edmund Ronald, Marc Schoenauer, and Dominique Snyers, editors, *Artificial Evolution*, **1363** of *Lecture Notes in Computer Science*, pages 275–285. Springer Berlin Heidelberg, 1998. 23, 45, 49

[106] G. Rudolph. Finite markov chain results in evolutionary computation: a tour d'horizon. *Fundam. Inf.*, **35**[1-4]:67–89, August 1998. 51

[107] P. V. Sander, D. Nehab, E. Chlamtac, and H. Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Trans. Graph.*, **27**[5]:144:1–144:9, December 2008. 126, 131

[108] K. Smyth, H. Hoos, and T. Stützle. Iterated robust tabu search for max-sat. In *Proceedings of the 16th Canadian society for computational studies of intelligence conference on Advances in artificial intelligence*, AI'03, pages 129–144, Berlin, Heidelberg, 2003. Springer-Verlag. 143

[109] P. F. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, editors, *Complex Systems and Binary Networks*, **461**, pages 77–163, Berlin, New York, 1995. Springer Verlag. 25, 96

[110] P. F. Stadler and G. P. Wagner. Algebraic theory of recombination spaces. *Evol. Comput.*, **5**[3]:241–275, 1997. 30

[111] D. Sudholt. General lower bounds for the running time of evolutionary algorithms. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, **6238** of *Lecture Notes in Computer Science*, pages 124–133. Springer Berlin Heidelberg, 2010. 52

[112] D. Sudholt. General Lower Bounds for the Running Time of Evolutionary Algorithms. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, **6238** of *Lecture Notes in Computer Science*, pages 124–133. Springer Berlin Heidelberg, 2010. 52

[113] D. Sudholt. Theory of swarm intelligence. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion '12, pages 1215–1238, New York, NY, USA, 2012. ACM. 115

[114] D. Thierens. An Adaptive Pursuit Strategy for Allocating Operator Probabilities. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1539–1546, New York, NY, USA, 2005. ACM. 43

[115] D. Thierens. Adaptive Strategies for Operator Allocation. In Fernando Lobo, Cláudio Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, **54** of *Studies in Computational Intelligence*, pages 77–90. Springer Berlin / Heidelberg, 2007. 43, 113

[116] A. Tuson and P. Ross. Adapting Operator Settings in Genetic Algorithms. *Evol. Comput.*, **6**[2]:161–184, June 1998. 3, 10, 38, 42, 43, 112, 113, 160

[117] L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Verel. Fitness Clouds and Problem Hardness in Genetic Programming. In *Genetic and Evolutionary Computation, GECCO 2004*, **3103** of *Lecture Notes in Computer Science*, pages 690–701. Springer, 2004. 60

[118] L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Vérel. Fitness Clouds and Problem Hardness in Genetic Programming. In *GECCO*, pages 690–701, 2004. 117

[119] L. Vanneschi, M. Tomassini, P. Collard, and S. Vérel. Negative Slope Coefficient: A Measure to Characterize Genetic Programming Fitness Landscapes. In *EuroGP*, pages 178–189, 2006. 4, 24, 25, 66, 67, 119

[120] V. K. Vassilev, T. C. Fogarty, and J. F. Miller. *Smoothness, Ruggedness and Neutrality of Fitness Landscapes: from Theory to Application*, pages 3–44. Springer-Verlag New York, Inc., New York, NY, USA, 2003. 4, 19

[121] S. Vérel, P. Collard, and M. Clergue. Where are bottlenecks in nk fitness landscapes? In *IEEE Congress on Evolutionary Computation*, pages 273–280, 2003. 19, 66

[122] M. D. Vose and A. H. Wright. Stability of vertex fixed points and applications. In *Foundations of Genetic Algorithms 3*, pages 103–113. Morgan Kaufmann, 1995. 19, 45, 49

[123] J. Watson, L. D. Whitley, and A. E. Howe. Linking search space structure, run-time dynamics, and problem difficulty: a step toward demystifying tabu search. *J. Artif. Int. Res.*, **24**[1]:221–261, August 2005. 143

[124] M. Wattenberg and A. Juels. Stochastic hill climbing as a baseline method for evaluating genetic algorithms. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1994. 102

[125] I. Wegener. Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In *Evolutionary Optimization*, **48** of *International Series in Operations Research and Management Science*, pages 349–369. Springer US, 2003. 51, 52, 115

[126] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, **63**[5]:325–336, 1990. 21, 30

[127] J. Whitacre, T. Pham, and R. Sarker. Credit Assignment in Adaptive Evolutionary Algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 1353–1360, New York, NY, USA, 2006. ACM. 43, 113

[128] J. M. Whitacre, T. Q. Pham, and R. A. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In *Proceedings of*

the 8th annual conference on Genetic and evolutionary computation, GECCO '06, pages 1345–1352, New York, NY, USA, 2006. ACM. 3, 42

[129] D. Whitley and A. M. Sutton. Partial neighborhoods of elementary landscapes. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 381–388, New York, NY, USA, 2009. ACM. 29, 99

[130] D. Whitley, A. M. Sutton, and A. E. Howe. Understanding elementary landscapes. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 585–592, New York, NY, USA, 2008. ACM. 26, 27, 28, 96, 97

[131] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, **1**[6]:80–83, 1945. 93, 135, 136, 139, 140, 141

[132] S. W. Wilson. Ga-easy doe not imply steepest-ascent optimizable. In *Proceedings of the 4th International Conference on Genetic Algorithms*, 1991. 19, 49

[133] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, **1**[1]:67–82, 1997. 3, 17

[134] S. Wright. The Roles of Mutation, Inbreeding, Crossbreeding, and Selection in Evolution. In *Proc. 6th Congr. Genetics*, **1**, page 365, 1932. 4, 17, 49

[135] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artif. Intell.*, **171**[8-9]:514–534, June 2007. 126, 131

[136] K. Xu and W. Li. Many hard examples in exact phase transitions. *Theor. Comput. Sci.*, **355**[3]:291–302, April 2006. 131

[137] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for sat. *J. Artif. Int. Res.*, **32**[1]:565–606, June 2008. 3, 7, 8, 32, 37, 38, 72, 158

[138] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *Evolutionary Computation, IEEE Transactions on*, **3**[2]:82–102, 1999. 3