

THE UNIVERSITY OF ADELAIDE

DOCTORAL THESIS

Diversity Optimization and Parameterized Analysis of Heuristic Search Methods for Combinatorial Optimization Problems

Author:
Wanru Gao

Principle Supervisor:
Prof. Frank Neumann

Co-Supervisor:
Dr. Markus Wagner

*A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy*

in the

Optimisation and Logistics
School of Computer Science

December 8, 2016

Declaration of Authorship

I, Wanru Gao, declare that this thesis titled, “Diversity Optimization and Parameterized Analysis of Heuristic Search Methods for Combinatorial Optimization Problems” and the work presented in it are my own.

- I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.
- I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.
- I also give permission for the digital version of my thesis to be made available on the web, via the University’s digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signed:

Date:

“Courage is not the absence of fear, but rather the assessment that something else is more important than fear.”

Franklin D. Roosevelt

THE UNIVERSITY OF ADELAIDE

Abstract

Faculty of Engineering, Computer & Mathematical Science

School of Computer Science

Doctor of Philosophy

Diversity Optimization and Parameterized Analysis of Heuristic Search Methods for Combinatorial Optimization Problems

by Wanru Gao

Heuristic search algorithms belong to the most successful approaches for many combinatorial optimization problems which have wide real world applications in various areas. The heuristic algorithms usually provide solutions with acceptable quality in reasonable time-frame which is different from exact algorithms. Fixed-parameter approach provides a way for understanding how and why heuristic methods perform well for prominent combinatorial optimization problems. In this thesis, there are two main topics discussed.

Firstly, we integrate the well-known branching approach for the classical combinatorial optimization problem, namely minimum vertex cover problem, to a local search algorithm and compare its performance with the core component of the state-of-the-art algorithm. After that, we investigate how well-performing local search algorithms for small or medium size instances can be scaled up to solve massive input instances. A parallel kernelization technique is proposed which is motivated by the assumption that huge graphs are composed of several easy to solve components while the overall problem is hard to solve.

Using evolutionary algorithms to generate a diverse set of solutions where all of them meet certain quality criteria has gained increasing interests in recent years. As the second section, we put forward an evolutionary algorithm which allows us to maximize the diversity over a set of solutions with good quality and then focus on the theoretical analysis of the algorithm to provide understanding of how evolutionary algorithms maximize the diversity of a population and guarantee the quality of all solutions at the same time. Then the idea is extended to evolving hard/easy optimization problem instances with diverse feature values. The feature-based analysis of heuristic search algorithms plays an important role in understanding the behaviour of the algorithm and our results show good classification of the problem instances in terms of hardness based on different combinations of feature values.

Acknowledgements

I would like to express my special appreciation and thanks to many people who have encouraged, inspired and helped me in the PhD study. Without their generous help, it will be impossible for this thesis to be finished. I would like to thank

- First of all, to my supervisor Frank Neumann. It has been an honour to be one of his PhD students. He has provided me great support, guidance and inspiration in doing research. I appreciate all his contributions of time, ideas and funding to make my PhD study inspiring and productive.
- To my co-supervisor Markus Wagner for his kind advices and encouragement in both researching and teaching.
- To all of the co-authors of the papers, Mojgan Pourhassan, Samadhi Nallaperuma, Benjamin Doerr, Tobias Friedrich and Timo Kötzing. I learned a lot in collaborations with others.
- To all the researchers and practitioners whose work is referred in my study for their great efforts in previous research. None of the studies is able to be conducted without the foundations set by other researchers.
- To all reviewers of the papers and the examiners of this thesis for their valuable comments.
- To the past and present group members in our research group that I have had the pleasure to work with, Bradley Alexander, Mingyu Guo, Sergey Polyakovskiy, Samadhi Nallaperuma, Mojgan Pourhassan, Shayan Poursoltan, Junhua Wu, Feng Shi, Aneta Neumann and other fellow students in the HDR office. It has been a great experience to work in the research group of Optimisation and Logistics in School of Computer Science. Sharing ideas in both research and daily life helps me progress a lot.
- To the School of Computer Science, Genetic and Evolutionary Computation Conference (GECCO) and the International Conference on Parallel Problem Solving from Nature (PPSN) for supporting my conference travel and research visit.
- To everyone in the group of Algorithm Engineering in Hasso Plattner Institute, Potsdam, Germany for sharing research ideas and providing feedback on my research during my stay with them.
- To all of the teachers who have taught or directed me in different areas for encouraging me to keep studying.
- Lastly, to my beloved family, especially my mom Jing and my dad Mingli, my relatives and friends for supporting me all the time in both life and research.

CONTENTS

Declaration of Authorship	i
Abstract	iii
Acknowledgements	iv
Contents	v
List of Figures	ix
List of Tables	xi
List of Abbreviations	xii
1 Introduction	1
2 Combinatorial Optimization and Heuristic Search	5
2.1 Introduction	5
2.2 Local Search	6
2.2.1 Metaheuristics for Local Search	6
2.3 Evolutionary Algorithms	8
2.3.1 General Issues of Evolutionary Algorithms	8
2.3.1.1 Representation	9
2.3.1.2 Variation Operator	9
2.3.1.3 Selection Mechanism	10
2.3.1.4 Other Issues	11
2.3.2 Main Types of Evolutionary Algorithms	11
2.3.2.1 Genetic Algorithms	11
2.3.2.2 Evolution Strategies	11
2.3.2.3 Evolutionary Programming	12
2.3.2.4 Genetic Programming	12

2.3.3	Evolutionary Multi-objective Optimization	12
2.4	Combinatorial Optimization	13
2.4.1	Travelling Salesman Problem	14
2.4.1.1	Exact Algorithms for TSP	15
2.4.1.2	Heuristics and Approximation Algorithms for TSP	16
2.4.2	Minimum Vertex Cover Problem	18
2.5	Diversity in Evolutionary Algorithm	19
2.5.1	Diversity in Objective Space	19
2.5.2	Diversity in Decision Space	20
2.5.3	Diversity Measurement	21
2.6	Conclusion	22
3	Algorithm Analysis and Problem Complexity in Evolutionary Algorithms	23
3.1	Introduction	23
3.2	Classical Computational Complexity Analysis	24
3.2.1	Fitness-based Partitions	24
3.2.2	Deviation Inequalities	25
3.2.3	Drift Analysis	27
3.3	Parameterized Analysis	28
3.3.1	Some Basic Definitions	28
3.3.2	Bounded Search Tree	29
3.3.3	Kernelization	30
3.4	Feature-based Analysis	30
3.4.1	Features Selection for Characterizing Problem Instances	31
3.4.2	Feature-based Analysis for Problem Hardness	32
3.5	Conclusion	33
4	Heuristic Algorithms for Minimum Vertex Cover Problem	34
4.1	Introduction	34
4.2	Background	35
4.3	Initialization Strategies	36
4.3.1	Different Initialization Approaches	36
4.3.2	Theoretical Analysis	38
4.3.3	Experimental Results	39
4.4	Two Local Search Algorithms for MVC	43
4.4.1	Edge-based Local Search Algorithm	43
4.4.2	Vertex-based Local Search Algorithm	44
4.4.3	Experimental Analysis	44
4.5	Conclusion	45
5	Scaling up Local Search for MVC in Massive Graphs	48
5.1	Introduction	48
5.2	Substructures in Massive Graphs	50
5.3	Parallel Kernelization for MVC	51
5.4	Experimental Results on DIMACS and BHOSLIB Benchmarks	52
5.4.1	Results for Original DIMACS and BHOSLIB Benchmarks	53
5.4.2	Results for Combined DIMACS and BHOSLIB Benchmarks	54
5.4.3	Results for Combination of Existing Hard Instances	55
5.5	Experimental Results on Real World Graphs	56
5.6	Remarks and Conclusions	59

6	Diversity Maximization for Single-objective Problems in Decision Space	60
6.1	Introduction	60
6.2	Background	61
6.2.1	Decision Space Diversity Measurement	61
6.2.2	Population-based Evolutionary Algorithm with Diversity Maximization	62
6.2.3	Classical Runtime Analysis Method for Evolutionary Algorithms	63
6.3	Diversity Maximization for OneMax Problem	64
6.3.1	OneMax Problem	64
6.3.2	Analysis of Large Threshold	66
6.3.3	Analysis of Smaller Threshold	69
6.4	Diversity Maximization LeadingOnes Problem	76
6.4.1	LeadingOnes Problem	76
6.4.2	Runtime Analysis for $(\mu + 1)$ -EA _D on LeadingOnes	77
6.5	Analysis of Vertex Cover Problem	80
6.5.1	Analysis of Complete Bipartite Graphs	81
6.5.1.1	$\epsilon < 1/3$	82
6.5.1.2	$\epsilon = 1/3$	83
6.5.1.3	$1/3 \leq \epsilon < 1/2$	85
6.5.2	Analysis of Paths	88
6.5.2.1	Paths with Even Number of Nodes	89
6.5.2.2	Paths with Odd Number of Nodes	92
6.6	Conclusion	93
7	Diversity Maximization for Multi-objective Problems in Decision Space	94
7.1	Introduction	94
7.2	Preliminaries	95
7.2.1	OneMinMax Problem	95
7.2.2	Hypervolumn-based Algorithm with Diversity Maximization	96
7.3	Search Space Diversity Optimization	98
7.3.1	Diversity Maximization for Multi-objective Problem	98
7.3.2	Experimental Analysis for Diversity Maximization for OneMinMax	99
7.3.3	Runtime Analysis for Diversity Maximization for OneMinMax	100
7.4	Conclusion	103
8	Feature-Based Diversity Optimization for Problem Instance Classification	104
8.1	Introduction	104
8.2	Preliminaries	105
8.2.1	Traveling Salesman Problem	105
8.2.2	Features of TSP Instance	106
8.2.3	Evolutionary Algorithm for Evolving Instance with Diversity Optimization	107
8.2.4	Experiments Setup	109
8.3	Range of Feature Values	109
8.4	Instance Classification Based on Multiple Features	113
8.4.1	Diversity Maximization over Single Feature Value	113
8.4.2	Diversity Maximization over Multiple Feature Values	116
8.5	Instance Classification Using Support Vector Machine	117
8.5.1	SVM with Linear Kernel	117
8.5.2	Nonlinear Classification with RBF Kernel	118
8.6	Diversity optimization for Instance Hardness	120
8.7	Conclusion	121

9 Conclusions	123
Bibliography	126

LIST OF FIGURES

2.1	The general procedure of an evolutionary algorithm expressed as a flow chart.	8
2.2	An example for a case that two individuals that are next to each other in objective space but are mapped to two points that are far away in decision space.	20
4.1	The histograms show the frequency that each algorithm gets the initial vertex cover of certain size for two sample instances from <i>DIMACS</i> benchmark set.	40
4.2	The histograms show the frequency that each algorithm gets the initial vertex cover of certain size for sample instances that randomly generated or from real world graph.	40
4.3	The box plots show the distribution of the results from 101 independent runs of seven different initialization approach on different minimum vertex cover instances.	42
4.4	The graphs shows the improvement of two local search algorithms in three example instances over iterations.	46
6.1	The $\mu \times n$ matrix represents the individuals in a population. In the example, it is a matrix for a population with 4 individuals which are all 8 bits in length. The 7th column is all-1-bit column and the 3rd column is 0-bit column as defined.	69
6.2	The example of global optimum for maximizing population diversity in One-Max where population size $\mu \leq \frac{n}{n-v}$.	71
6.3	The $\mu \times n$ matrix represents the solutions of MVC in a population. In the example, each row represent an individual. In each row, the left ϵn bits and right $(1 - \epsilon)n$ represent the existence of nodes from set V_1 and set V_2 in the individual respectively.	81
6.4	When $\epsilon = 1/3$, the population of MVC with maximized diversity should include cover sets with different types.	84
6.5	In the complete bipartite graph with large ϵ , it is possible at some stage, the last individual of one type is replaced by an individual of the other type.	87

6.6	Arrange all $(k + 1)$ optimal solutions for the MVC problem of a path with $n = 2k$ according to the ascending order of numbers of '01' pairs.	89
7.1	The 8×7 matrices represent some populations for which there is no 1-bit flip can improve the population diversity. The last rows report the numbers of 1-bits in corresponding columns.	100
8.1	The boxplots show the distribution of some feature values of a population consisting of 100 different hard or easy TSP instances of different number of cities without or with diversity mechanism.	110
8.2	Some examples of the evolved hard TSP instances of different number of cities are shown with an optimal tour computed by Concorde.	112
8.3	2D Plots of feature combinations which provide a separation between easy and hard instances. The blue dots and orange dots represent hard and easy instances respectively.	113
8.4	2D Plots of feature combinations which does not provide a clear separation between easy and hard instances. The blue dots and orange dots represent hard and easy instances respectively.	113
8.5	3D Plots of combining experiment results from maximizing the diversity over features <i>mst_dists_mean</i> , <i>nnds_mean</i> and <i>chull_area</i> , which provides a good separation of easy and hard instances.	115
8.6	3D Plots of combining experiment results from maximizing the diversity over features <i>mst_dists_mean</i> , <i>chull_area</i> and <i>centroid_mean_distance_to_centroid</i> , which provides a good separation of easy and hard instances.	115
8.7	3D Plots of combining experiment results from maximizing the diversity over features <i>mst_dists_mean</i> , <i>chull_area</i> and <i>mst_depth_mean</i> , which provides a good separation of easy and hard instances.	115
8.8	3D Plots of combining experiment results from maximizing the diversity over features <i>mst_dists_mean</i> , <i>nnds_mean</i> and <i>chull_area</i> with considering of weighting, which provides a good separation of easy and hard instances. Legend is the same as that in Figure 8.5.	116
8.9	3D Plots of experiment results from maximizing the diversity over approximation ratio in the feature space of feature combination <i>chull_area</i> , <i>mst_dist_mean</i> and <i>mst_depth_mean</i> . The color of the dot reflect the hardness of the problem.	121
8.10	3D Plots of experiment results from maximizing the diversity over approximation ratio in the feature space of feature combination <i>chull_area</i> , <i>angle_mean</i> and <i>cluster_10pct_mean_distance_to_centroid</i> . The color of the dot reflect the hardness of the problem.	121

LIST OF TABLES

4.1	Experimental results on instances comparing the statistics between Algorithm 4.1 and 4.2.	41
4.2	Performance comparison between Algorithm 4.5 and 4.6 on some sample instances. The average vertex cover size is listed after running each algorithm for certain number of iterations.	46
5.1	This table contains the results from NuMVC-PK and NuMVC on the BHOSLIB and DIMACS benchmark.	53
5.2	This table contains instances that have been tested on, which are generated by duplicating one existing hard instance in BHOSLIB benchmark.	54
5.3	This table contains instances that have been tested on, which are generated by combining several existing hard instance in BHOSLIB benchmark.	55
5.4	Experimental results of NuMVC-PK and single run of NuMVC on instances from some real world graphs about social networks.	57
5.5	Experimental results of NuMVC-PK and single run of NuMVC on instances from some real world graphs about collaboration networks.	57
5.6	Experimental results on instances from some real world graphs with different parameter settings. $\mu = 3$ in all cases.	58
5.7	Experimental results on instances from some duplicated benchmark graphs with different parameter settings. $\mu = 3$ in all cases.	59
8.1	This table lists the accuracy of SVM with RBF kernel separating the hard and easy instances in 21 different two-feature space.	119
8.2	This table lists the accuracy of SVM with RBF kernel separating the hard and easy instances in 35 different three-feature space.	119

LIST OF ABBREVIATIONS

OMM	One Min Max
RLS	Randomized Local Search
EA	Evolutionary Algorithm
GA	Genetic Algorithm
NP	Nondeterministic Polynomial
TSP	Traveling Salesman Problem
MVC	Minimum Vertex Cover
MOP	Multi-objective Optimization Problem
MOEA	Multi-Objective Evolutionary Algorithm
w.l.o.g.	without loss of generality
i.e.	id est (that is)
e.g.	exempli gratia (for example)

To my beloved parents...

Learn from yesterday, live for today, hope for tomorrow.

Albert Einstein

CHAPTER 1

INTRODUCTION

In applied mathematics and theoretical computer science, combinatorial optimization is a popular topic aiming at finding the optimal solution from a finite set, which integrates techniques from many areas such as combinatorics, linear programming and algorithm theory [26]. Combinatorial optimization has many applications in various fields, such as artificial intelligence, machine learning, software engineering and mathematics. Two of the well-known problems involving combinatorial optimization are the Travelling Salesman Problem (TSP) and Minimum Vertex Cover (MVC) problem, which both come from generalization forms of Karp's 21 NP-complete problems [89].

The TSP is one of the famous combinatorial optimization problems, which plays an important role in both practical application and theoretical research [5]. Given a set of cities and the distances between each pair of cities, the goal of TSP is to compute a tour of minimal length. The tour should guarantee that each city is visited exactly once and returns to the origin at the end.

The MVC problem is another well-known NP-hard combinatorial optimization problem [126] which has many real-world applications and is regarded as an important sample problem in the theoretical analysis. Given an undirected graph, the target of the MVC problem is to find the smallest subset of the vertex set such that for each edge at least one of its endpoints is in the set.

Since both problems are NP-hard [89, 126], there does not exist any algorithm that is able to optimize the problems in polynomial time with $P \neq NP$. After many years' research,

researchers have found that heuristic search approaches, such as local search, simulated annealing, evolutionary algorithms and ant colony optimization, perform well in solving various combinatorial optimization problems [25, 1, 78]. Although heuristic algorithms are not able to guarantee the solution to be optimal, they usually provide nearly optimal solutions in reasonable computational time. There have been many different heuristic algorithms designed to solve the TSP and the MVC problem. Different from the exact algorithms which guarantee the optimality of solutions but suffer from the exponential worst-case execution time, the heuristic approaches can be terminated after some time and return solution(s) with acceptable quality.

Although there has been rapid development in the area of experimental research and application of heuristic search, the theoretical analysis is still left far away. The randomized search heuristics inspired by nature are hard to analyse because of the randomness in the algorithm. However, the theoretical analysis lies in an important position in understanding the characteristics and behaviours of the algorithms [126]. In the past few decades, many researchers have devoted themselves into improving the theoretical understanding of the algorithms. As the initial step, many studies focus on simple example functions [42, 69, 138, 160], which then lead to the analyses about combinatorial optimization problems [118, 122, 104]. The classical computational complexity analyses often focus on the worst-case expected time. This approach aims at finding the expected runtime until the global optimal solution is found and is beneficial for understanding algorithms which can solve certain problem in polynomial time. Useful tools for the complexity analysis of algorithm include fitness-based partitions [159], deviation inequalities [112] and drift analyses [69, 68, 38]. The parameterized analysis of heuristic search approaches has gained more and more attention during the past few years [150, 94, 28, 95, 114]. In parameterized analysis, the source of exponential complexity in the problem is often separated to enable the analysis based on problem structure [41]. It provides a way to understand the behaviours of heuristic methods in solving combinatorial optimization problems. The feature-based analysis is different from the classical algorithm analysis which often takes a worst case perspective, it allows us to examine the problem based on its features and characteristics. It can be seen as an important mechanism which bridges the gap between pure experimental investigations and mathematical methods for the performance analysis of heuristic search algorithms [119, 91, 44].

We carry on research into the design of heuristic search algorithms for the sample combinatorial optimization problem MVC and investigation of different approaches from the theoretical aspect. The proposed approach can be extended to other combinatorial optimization problems.

Evolutionary algorithms belong to a type of heuristic search algorithms which have wide applications in many areas. Evolutionary algorithms usually work with a set of solutions which is called a population. Diversity mechanisms can be regarded as the key to the working behaviours of population-based algorithms [17, 157]. A diverse population is one of the key factors to guarantee the coverage of explored solutions in the whole space. From

the optimization point of view, a diverse set of individuals in a population can contribute to preventing the algorithm from premature converging to the local optima. On the other hand, from the design point of view, decision makers are given different choices with the diverse set of solutions. The benefit in obtaining such a set of solutions lies in the variety of solutions and the intuitive knowledge of decision makers such as engineers to incorporate their personal knowledge when picking a solution for the final implementation.

The idea of using evolutionary approaches to obtain search space diversity under the condition that each of the solutions has at least certain quality has recently been proposed in the context of single- and multi-objective optimization [156, 154, 155]. In contrast to the common approach of using diversity to improve the quality of a single solution, the goal of these studies focuses on achieving a diverse set of solutions which all fulfil certain quality requirements. In our study, we follow this idea of generating diverse sets of solutions using EAs and conduct investigation on different optimization problems in both theoretical and practical perspectives.

This thesis consists of the following three main parts, which are introduction and background for the research discussed in the thesis, heuristic algorithms for combinatorial optimization and diversity optimization in decision space and feature space. The rest of this thesis is organized as follows.

Chapter 2 and 3 provide a brief introduction and background about the studies conducted and discussed in this thesis. In Chapter 2, two well-known combinatorial optimization problems, namely the travelling salesman problem and the minimum vertex cover problem, are introduced with problem setting, formulation and different algorithms. The background knowledge about evolutionary algorithms is included which sets the basis for later discussion. The last section of Chapter 2 focuses on the diversity in evolutionary algorithms which will be discussed in details in the third part of the thesis. Chapter 3 consists of an introduction about different algorithm analysis mechanisms.

Chapter 4 and 5 mainly focus on our research into heuristic search approaches for solving the MVC problem. We introduce two parameterized local search algorithms for the MVC problem in Chapter 4 and both of these algorithms are based on the bounded search tree approach. We examine the different performance of different initialization approaches for the MVC problem and details are included in this chapter. With the comparison between the fixed parameter branching algorithm and the edge-based local search algorithm which is equivalent to the core component of the state-of-the-art algorithm, we contribute to the incorporation of the more complex vertex-based branching rules to the MVC algorithm and show that this leads to better results in many test cases. This chapter is based on a conference paper published in the International Conference on Parallel Problem Solving from Nature (PPSN) [51]. Then in Chapter 5 we propose an approach for solving huge MVC problems with an existing solver. The main idea is named as parallel kernelization and based on the assumption that large graph is composed of several easy-solved partitions. The kernelization approach can be applied in solving many combinatorial optimization problems and

with the MVC problem as an example we show that it has good performance in practice.

The following chapters focus on the diversity optimization in EAs. In Chapter 6, a theoretical analysis is included for decision space population diversity maximization on some sample single-objective optimization problems. The main contribution of this section is that it provide some insights into the theoretical understanding of diversity mechanisms used in evolutionary algorithms by means of rigorous runtime analyses. We also propose an evolutionary algorithm that is able to generate a diverse set of solutions in decision space which are all of good quality. This chapter is extended from the work published at Genetic and Evolutionary Computation Conference (GECCO) [53, 54]. The runtime analysis for multi-objective optimization problem is extended from the analysis of single-objective problem and is detailed in Chapter 7. Some initial theoretical investigations into the sample multi-objective optimization problem are described in this chapter in details. After the theoretical analysis sets the foundation, our research moves to the application domain. The contents are based on the runtime analysis results published at GECCO [37]. In Chapter 8, we conduct investigations into diversity maximization in feature space for the sample problem of TSP. We propose a new approach of constructing hard and easy instances for a certain combinatorial optimization problem. Following the idea of generating diverse sets of instances which are all of high quality in Chapter 6, we introduce an EA which maximize diversity of the obtained instances in terms of features. The proposed approach has been shown to give a much better classification of instances according to their difficulty in being solved by a certain algorithm under investigation. This chapter can be seen as an extension of the results published in the conference PPSN [52].

Chapter 9 concludes the whole thesis with a brief summary of the research and some remarks on any potential future research directions.

I hear and I forget. I see and I remember. I do and I understand.

Xuncius

CHAPTER 2

COMBINATORIAL OPTIMIZATION AND HEURISTIC SEARCH

2.1 Introduction

Combinatorial optimization is a broad field in applied mathematics and theoretical computer science, which integrates techniques from combinatorics, linear programming and algorithm theory [26]. Combinatorial optimization problems aim at finding one or more optimal solutions in a well-defined discrete problem space with certain constraints. Because of its success in solving difficult problems in many application areas, such as telecommunication, task allocation and schedule design, this field has attracted more and more attention and interests [119, 127].

Heuristic search methods such as local search, simulated annealing, evolutionary algorithms and ant colony optimization, have been proved suitable for solving various combinatorial optimization problems [1, 78]. A heuristic aims at producing a solution in reasonable time and the solution found is with acceptable quality. Although heuristic algorithms are usually not able to guarantee the optimality of the solution, they can find nearly optimal solutions within reasonable timeframe.

This chapter provides a general overview of the combinatorial optimization by introducing some classical problems. In the second part of this chapter, we include a discussion about the heuristic algorithms for different problems. This chapter is organized as follows, some

background knowledge about local search and evolutionary algorithms is included in Section 2.2 and Section 2.3. In Section 2.4, two classical combinatorial optimization problems are introduced, namely Travelling Salesman Problem and Minimum Vertex Cover problem, with problem formulation and some popular algorithms. Then in Section 2.5, we introduce the diversity in objective space and decision space and its importance in evolutionary algorithms.

2.2 Local Search

Local search is a type of heuristic methods which is often used to solve computationally hard optimization problems [1]. Different from global search, local search focuses on exploring the local neighbourhood rather than the whole search space. It aims at finding the good solution in reasonable timeframe. Although local search heuristics cannot guarantee the global optimality of the solution found, they have been shown to be very successful in dealing with large difficult optimization problems with consideration of the trade-off between computational complexity and the solution quality. Local search has attracted more and more attention of the researchers and practitioners because of its efficiency, flexibility and the property that easier to understand and implement than its exact counterpart.

In local search heuristics, the starting point is a feasible solution that may be generated by different methods. In each iteration, one of the solutions from the neighbourhood of the current solution is selected. Typically, there are more than one neighbour of the current solution and decision needs to be made on which one to move to. Even if interrupted before termination, local search still returns a feasible solution. The basic idea of local search algorithm is the iterative exploration of the neighbourhood of the current best solution and improvement by simple local changes. The definition of neighbourhood is crucial which restricts the possible solutions that can be reached from the current solution in one single step of a local search algorithm.

2.2.1 Metaheuristics for Local Search

Metaheuristics are high-level general-purpose strategies that provide guidance for problem specific heuristics. A metaheuristic is formally defined as a process that iteratively generate solutions which structure information to find near-optimal solutions [124]. Metaheuristics are strategies that provide guidance for the search process [13]. They are approximate and usually non-deterministic.

One fundamental metaheuristic that belongs to local search is hill climbing. It starts with an arbitrary solution and iteratively improves the current solution by incremental changes to the solution. The termination criterion is when no further improvement can be found in the neighbourhood. Greedy local search is a form of hill climbing search in which the local change leads to the largest improvement of fitness function is selected. Hill climbing

is good at finding local optimal solutions and can be applied to many combinatorial optimization problems, e. g. Lin-Kernighan algorithm is able to find good solutions for large TSP instances by local changes to the current shortest tour [101].

The general problem of local search approaches is that they may get stuck in local optima where no improving neighbours are available. Since local search cannot be used to determine whether the best solution found so far is globally optimal or not, local search may terminate with the locally optimal solution. This problem can be overcome by restarts or other complex schemes, such as simulated annealing [90] and iterated local search [103]. There have been many metaheuristics with improved performance that are inspired by the nature.

Simulated annealing is one example of local search that accepts 'downhill' moves which means moves lead to solutions with objective value the same as or smaller than that of the current best solution. As its name indicates, simulated annealing is inspired by the physical process of annealing and has many successful applications [90, 92]. In simulated annealing, 'downhill' moves are accepted with a probability based on the change in the fitness value. Additionally, there is another parameter called 'temperature' in simulated annealing that controls the acceptance rate of the neighbours. When the temperature parameter goes down, the probability of accepting worse solutions decreases. When the temperature is high, the algorithm has a higher probability to explore wider solution space.

Iterated local search also offers a way to leave the local optima by changing the current solution through other rules which lead to new solutions beyond the neighbourhood [103]. The iterated local search is expected to perform better than a simple restart of the local search which implies the information obtained before is discarded.

Another example that relaxes the local search rule to get away from the local optima is tabu search [57]. In tabu search, a list that records the recently visited solutions is introduced in order to prevent the algorithm from going back to the previously visited solutions for a limited period of time. Moreover, the 'downhill' moves have possibilities to be accepted if there is no improving move available in the current iteration. The implementation of tabu search makes use of memory structure so that the algorithm does not go back to the same situation repeatedly.

Ant colony optimization is a recently developed population-based approach which is inspired by the foraging behaviour of ants [39]. The ants exchange information with each other via pheromones and therefore leave paths between their colony and some location of food. The ants following the pheromone trail leave pheromones as well and gives positive feedback. The algorithm simulates a set of artificial ants and constructs the solution based on the nature of ants in each iteration.

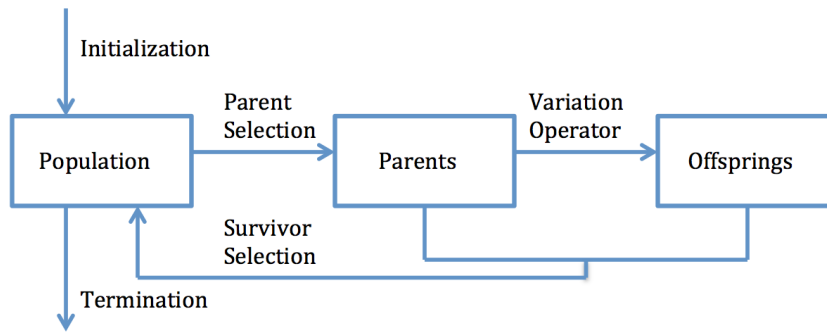


FIGURE 2.1: The general procedure of an evolutionary algorithm expressed as a flow chart.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are population-based metaheuristic optimization algorithms which are inspired by the natural selection (survival of the fittest) [43, 111]. Given a quality evaluation function, some of the better candidates are chosen to seed the next generation by generating new offsprings. The common underlying idea of EAs can be summarized as follows. Given a population of candidate solutions and a quality evaluation function, the pressure from the environment causes natural selection inside the population that includes new offsprings, which results in the improvement in the fitness of the whole population [43].

EAs have become popular since the mid-1960s and different approaches have been proposed in the last few decades [119]. EAs have shown good performance in providing approximation solutions to problems arise in many different areas, such as engineering, arts, economics and sciences [106]. In this section, we give a brief overview about EA and some main paradigms of evolutionary computation.

2.3.1 General Issues of Evolutionary Algorithms

The general structure of an evolutionary computation algorithm is simple, as shown in Figure 2.1. The main difference between EAs and local search is that EAs usually work with a set of solutions in each iteration [119]. This set of candidate solutions is called the population of an EA. An EA maintains a set of individuals for a certain iteration. Each individual refers to a candidate solution to the optimization problem and is represented in some data structure. After an initialization process, a population is generated to be worked on in the further steps. In the selection process, the solutions are evaluated based on a measurement function which is named as *fitness*. A selector is used to choose a set of individuals as the parent population. Variation operators are used to generate new individuals to form the offspring set. As the last step in each iteration, all potential solutions in the current population are filtrated by the survivor selector to compose the population for the next iteration. The whole process is repeated until termination criterion is met.

The procedure of EAs is summarized as above and there are some important components which need to be specified in order to define an EA.

2.3.1.1 Representation

How to relate the original problem to some concepts in computer science is very important. The first issue needs to be settled down in most algorithms is to abstract the real world problem. It is the same in defining an EA. Representation, which refers to the definition of individuals, is the first thing needs to be cared about. For successful and efficient use of an EA, defining a proper representation is essential and appropriate search operator design will also benefit from it [137].

There are many different ways in which solutions can be represented. The decision about individual definition should be made based on the problem and some other issues including memory consumption, fitness calculation and variation operation should also be taken into consideration. A good representation gives a good abstraction of the useful information about the candidate solution in real world, for example, an ordered list with each element stores the index for a city and each element appears exactly once can be used to represent a possible tour for a TSP. A binary string is suitable for the representation of an individual in the MVC problem but may be a poor representation for the TSP, since it is hard to map a bitstring to a permutation of cities.

In an EA, the population refers to a set of possible solutions. The formation of a population changes over iteration. Typically the population is defined as a fixed size multiset of individuals. In some sophisticated EAs, a population may have additional spatial structure, such as distance or diversity measurement which will be discussed in later section in this chapter.

2.3.1.2 Variation Operator

Variation operators are important in introducing new solutions to the current population. The variation operators are designed to work on old individuals to produce new ones, therefore they should fit the chosen representation in order to work properly. There are two types of variation operators, which are mutation and crossover operators.

A mutation operator works on a single individual and produces a modified offspring from it. A mutation operator is often designed as dependent on probabilities. It performs different roles in different EA types. In genetic algorithms, mutators are used to explore the area in the search space that has not been reached yet. One example of mutation operators on binary strings which has been widely used in theoretical analysis in EA is introduced as follows. In a bitstring of length n , a mutator can be defined as flipping each bit with a certain probability p independently from each other. In order to prevent the operator from obtaining an offspring like generating a random bitstring from scratch, the probability p

should not be too large. One common choice of p is $1/n$, which implies one bit is flipped on average.

A recombination or crossover operator takes at least two parent individuals and produces new solutions based on them. Similar to mutation operators, crossover operators involve probability as part of the change as well. The crossover operator extracts information from the parents and merges the information to form offsprings. Both the ways to extract information and to merge information may be controlled by probabilities. By mating two or more individuals with different but desirable features, recombination operator may be able to produce offsprings that combine these features. In an example crossover operator for binary string, a crossover point is selected randomly. The data beyond the certain point in the two parent binary strings is swapped to form two offsprings. This crossover operator is called one-point crossover and the feasibility of the resulting offsprings needs to be checked.

2.3.1.3 Selection Mechanism

Before introducing the selection mechanism, the evaluation function of the individuals should be defined first. The evaluation function is usually called the fitness function which allows the algorithm to quantify the requirements and thereby sets the basis for selection and facilitate the algorithm. The evaluation function defines what improvement means in the specific optimization problem. A proper defined evaluation function is able to measure the quality of candidate solutions and accelerate the process of improvement.

There are two places where selection mechanisms may get involved in an EA, which are parent selection and survivor selection. The selectors in EA are used to decide the choice of individuals based on their quality. The aim of parent selector is to select individuals from the current population to generate offsprings. The parent selection is typically probabilistic and the individual with higher quality is usually given higher probabilities to be selected. By doing this, the better solutions have higher probabilities to become parents and pass their good features to the next generation. The survivor selection is used to decide on the individuals from both the parent and offspring population to form the population in the next generation. Since in EAs the population size is typically constant, the individuals with better quality, which can be measured explicitly by the fitness function, should be given higher probability to stay in the population for the next generation.

There are many different selection methods. Fitness proportional selection gives probability for an individual to stay proportional to the fitness of the individual. In the tournament selection, some pre-defined number of individuals compete for selection based on their fitness values. $(\mu + \lambda)$ and (μ, λ) -selection are two selection mechanisms of great importance in both practical and theoretical aspects. The difference between them lies in the survivor selection methods and will be discussed in more details in the following section.

2.3.1.4 Other Issues

The initialization process of an EA is usually very simple. In some cases, a population of randomly generated individuals can be seen as a good start for an EA.

The termination criteria for EAs mainly fall into the following aspects: the total number of generations, the overall running time of the algorithm and the quality of the solutions. Since EAs are stochastic and there is no guarantee for them to reach the global optimal fitness, the quality requirement for termination can be set to some nearly optimal values.

2.3.2 Main Types of Evolutionary Algorithms

There are a few main types of evolutionary computation techniques including genetic algorithms, evolution strategies, evolutionary programming and genetic programming. In this section, we give a brief introduction of them. A detailed discussion about different EAs can be found in [43].

2.3.2.1 Genetic Algorithms

Genetic Algorithms (GAs) were introduced by Holland in [76] as a method of studying adaptive behaviour. GAs mainly work in discrete search space and usually use binary strings as representation. Other popular representations include real-valued vectors and permutation representation. In GAs, recombination undertakes the main workload for generating good offsprings from the current population. Mutation operators are regarded as minor variation operators and often applied to the offsprings produced by crossover with a low probability. The commonly used selection method for both parent and survivor selection in GAs is the fitness proportional selection. GAs are simple to implement but it is hard to understand their behaviours. In [58], Goldberg gives some insights into the theoretical understanding of the heuristics.

2.3.2.2 Evolution Strategies

Evolution Strategies (ESs) were proposed in early 1960s and further developed in the 1970s [131], which are used to solve continuous optimization problems and parameter optimization problems. In ESs, the individuals are usually represented by real-valued vectors and mutation is mainly used as the variation operator. The survivor selection in ESs is deterministic and based on the fitness rankings. There are two important strategies which are $(\mu + \lambda)$ -ES and (μ, λ) -ES. In $(\mu + \lambda)$ -ES, in each iteration λ offsprings are produced and together with the μ individuals in the current population form the pool of individuals to be selected from. The population for next generation is selected from the $(\mu + \lambda)$ individuals based on their fitness values. In the case of (μ, λ) -ES, λ offsprings are generated from the parent population as in $(\mu + \lambda)$ -ES, whereas the individuals in the next generation's population are only

selected from the λ offsprings, which implies $\lambda \gg \mu$. The special case of $(\mu + \lambda)$ -ES when $\lambda = 1$ is often investigated theoretically as the initial step in studies into population-based algorithms.

2.3.2.3 Evolutionary Programming

Evolutionary Programming (EP) techniques were introduced by Lawrence Fogel [47]. EP aims at achieving a sequence of symbols that predicts the problem in a more accurate way. One commonly used representation for EP is finite state machine. EP usually only use mutation as the variation operator and the population for the next generation is selected with a probabilistic selection method. In the survivor selection phase of EP, elitism is often used to guarantee the best solution found so far stays in the population for the next generation.

2.3.2.4 Genetic Programming

Genetic Programming (GP) introduced by Koza [93] is a technique used to construct computer programs to solve some given tasks. Instead of designing an evolution program to solve a problem, GP searches the possible computer programs for the fittest one in solving the problem. The individuals for GP are computer programs which are usually represented as trees. The fitness of a program is evaluated by its performance on some test cases. In each iteration, new program is generated by applying crossover and mutation operators and usually the fitness proportional selection is used to select the population for the next generation.

2.3.3 Evolutionary Multi-objective Optimization

In multi-criterion optimization, the optimization problem involves a set of objective functions, which need to be optimized simultaneously but may be conflict to each other [169]. Multi-objective optimization has many applications in different areas including engineering [24, 107], economics [151] and scheduling [98].

For a Multi-objective Optimization Problem (MOP), the aim is to make an optimal decision based on trade-offs between all objectives. Therefore, there is no single optimal solution for a MOP. A candidate solution is said to be Pareto optimal for a MOP if these does not exist any other feasible solutions that will decrease in some objective values without resulting in a simultaneous increase in at least one other objective value. The Pareto set refers to a set of solutions that contains all Pareto optimal solutions.

Assume there are k objective functions $f_i : X \rightarrow \mathbb{R}, 1 \leq i \leq k$ which map a solution x in decision space X to an objective vector $f(x) = (f_1(x), f_2(x), \dots, f_k(x))$ in the objective space $\mathbb{R}^k$. The k objective functions all need to be maximized. Then the *Pareto dominance* is defined as follows.

Definition 2.1. A solution $x \in X$ is defined to dominate another solution $y \in X$ iff $\forall i \in [1, k]$, it fulfils $f_i(x) \leq f_i(y)$ and $\exists i \in [1, k]$, that $f_i(x) < f_i(y)$.

With this definition, the *Pareto-optimal* and *Pareto set* are defined. And the graph of the Pareto set in objective space is called the *Pareto front*.

Definition 2.2. A solution $x' \in X$ is defined as *Pareto-optimal* iff there is no other solution in X that dominates it.

The target of Multi-Objective Evolutionary Algorithms (MOEAs) is to achieve a set of solutions that gives a better coverage of the Pareto front, which is first introduced in 1985 by Schaffer [143]. After that, many MOEAs have been proposed and some of them have shown good performance in solving different MOPs. There are a few algorithms designed based on the Pareto dominance. Some of the well-known MOEAs include NSGA-II [35], SPEA2 [170] and MOEA/D [166].

For assessing the quality of the solution set found by MOEAs, it is necessary to have some evaluation functions since unlike that in single-objective optimization, it is not easy to compare different sets in multi-objective optimization. The common ways to deal with this problem are attainment functions and quality indicators. The attainment function approach can be seen as a general form of the cumulative distribution function $F_X(z) = P(X \leq z)$ where X denotes a real-value random variable and $z \in \mathbb{R}$. The quality indicators are functions that map a Pareto set approximations to a real value. The ϵ -indicator and hypervolume indicator are two of the well-applied indicators which have been shown to be able to lead to good results in practice [168, 12].

The hypervolume indicator measures a set of elements in $\mathbb{R}^m$ (corresponding to images of elements in S) with the volume of the dominated portion of the objective space. Assume that the fitness function is defined as $f(x) = (f_1(x), f_2(x), \dots, f_m(x))$. Formally, given a reference point $r \in \mathbb{R}^m$, the hypervolume indicator can be defined for a given set $A \subset X$ as

$$I_H(A) = \lambda \left(\bigcup_{a \in A} [f_1(a), r_1] \times [f_2(a), r_2] \times \dots \times [f_m(a), r_m] \right).$$

The contribution of individual x to the hypervolume indicator of population P is defined as

$$c_H(x) = I_H(P) - I_H(P \setminus \{x\}).$$

2.4 Combinatorial Optimization

There are many real world problems which can be abstracted as combinatorial optimization problems. A *combinatorial optimization problem* can be represented formally as a triple (S, f, Ω) , where S denotes a well-defined search space, f denotes the objective function and Ω denotes a set of constraints. The goal of combinatorial optimization is to find one or more

globally optimal solutions according to the fitness function. There are many example combinatorial optimization problems that have been studied intensively, such as minimum spanning tree problem, knapsack problem, integer programming and travelling salesman problem.

Throughout this thesis, we focus on the combinatorial optimization problems in the search space of $\mathbb{R}^n$. The goal is to find a solution that has the highest (or smallest) objective value and fulfils all constraints.

In this section, we give an introduction of two well-known combinatorial optimization problems, which are the Travelling Salesman Problem (TSP) and Minimum Vertex Cover (MVC) problem. Both of the two problems are based on graphs.

2.4.1 Travelling Salesman Problem

The *Travelling Salesman Problem* is one of the famous NP-hard combinatorial optimization problems which has great importance in both practical and theoretical aspects. Given a set of cities and the distances between each pair of cities, the aim of TSP is to find a shortest possible route which visits each city exactly once and returns to the origin city at the end. The general form of the TSP has firstly been studied by mathematician Karl Menger in the 1930s. The problem has been investigated intensively in both theoretical and practical studies [5] and proved to be NP-complete in [89].

The TSP has many real-world applications in different areas. With its original formulation, it has been applied to scheduling, logistics and manufacturing fields. A direct application of the TSP in the drilling problem of printed circuit boards has been introduced in [63]. The school bus routing problem is found to be solvable by modelling as the TSP [3]. The design of global navigation satellite system surveying network can also benefit from applying the TSP [141]. After some modification or transformation, the TSP can also be treated as a part of other problems such as genome sequencing [2], crop surveys [161] and spacecraft interferometry [22].

There are different types of TSP which can be applied in different areas according to their properties. The TSP instances mainly fall into three different categories, which are *asymmetric TSP*, *symmetric TSP* and *multi-TSP*. Most TSP instances are symmetric TSP instances where the distances between two cities are the same in both direction. There exists another type of TSP which is asymmetric TSP in which the distance from city x to city y is not equal to the distance from y to x . As its name indicates, multi-TSP involves multiple salesmen and these salesmen may finish their tours in different cities based on the definition of the problem.

Some widely studied problem types are *metric TSP*, *Euclidean TSP* and *Manhattan TSP*. In the metric TSP, the distance between each pair of cities satisfies the triangle inequality. Both Euclidean TSP and Manhattan TSP are metric TSP. The difference between these two types of

TSP instances is the function used to calculate the distances between pairs of cities. In the Euclidean TSP, the cities are represented by points in the Euclidean plane and the distance between two cities is the Euclidean distance which is formulated as $d(x, y) = \sqrt{\sum_{i=1}^{dim} (x_i - y_i)^2}$, where dim represents the dimension of the space. In Manhattan TSP, the distance between two cities refers to the sum of the differences in each coordinate.

In this thesis, we focus on the investigation into the Euclidean TSP, which is metric and symmetric.

In general, the TSP can be formulated as follows. Given a set $V = \{v_1, v_2, \dots, v_n\}$ of n cities and a distance function $d : V \times V \rightarrow \mathbb{R}_{\geq 0}$, the task of TSP is to compute a shortest route that visits each city v_i exactly once and returns to the origin. Represented as an undirected graph $G = (V, E)$ that has a non-negative integer cost $c(u, v)$ associated with each edge $(u, v) \in E$, the TSP aims at finding the Hamiltonian cycle with the minimum total cost. A Hamiltonian cycle refers to a cycle in the graph that visits each vertex exactly once.

A solution to the TSP is often represented by a permutation $\pi = (\pi_1, \dots, \pi_n)$ of the n cities where π_i denotes the i th city in the city list and the fitness function to be minimized in this problem is the total tour length which can be represented as

$$c(\pi) = \sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1}) + d(\pi_n, \pi_1).$$

TSP has been studied for years and there have been many algorithms proposed. The algorithms for the TSP mainly fall into two categories, which are exact algorithms and approximation algorithms.

2.4.1.1 Exact Algorithms for TSP

There have been a large number of exact algorithms proposed for the TSP. They are designed to find the optimal solution to the TSP. The exact algorithms are usually computationally expensive since they need to consider all solutions either explicitly or implicitly in order to guarantee the optimality. The first and simplest approach would be going through every permutation of the cities and comparing the cost of each tour, which can be seen as a brute force search. The expected runtime for this approach is $O(n!)$, where n denotes the number of cities. The brute force search is very time consuming, especially when the number of cities goes up.

The Bellman-Held-Karp algorithm [11, 72] is an application of dynamic programming in solving the TSP. This approach takes expected $O(2^n n^2)$ time to solve a TSP instance with n cities.

Some of the exact algorithms can be better understood and explained in the context of integer linear programming [96]. In the work of Dantzig, Fulkerson and Johnson [32], the integer linear programming formulation of the TSP is introduced for the first time. The

model is usually relaxed and then solved using linear programming techniques. In [32], a case study of instances with 49 cities, which is a relatively large size at that time, is conducted. These studies set the foundation for cutting-plane algorithms of the TSP and the algorithms are further discussed in [86].

The branch and bound algorithms are also used to find an optimal solution to the TSP and based on the relaxation of the integer program. The candidate solutions are considered as leaves of a rooted tree and the algorithm recursively explores the branches of the tree which represent subsets of the candidate solution set. A branch is discarded according to the branching rules. This approach is introduced to solve the TSP [82].

The branch and cut algorithm is a combination of the cutting plan algorithm and the branch and bound algorithm [125]. When there is no optimal solution to the integer program found by using the cutting plane mechanism, the algorithm branches to the next stage. This algorithm is the state-of-the-art solution for large instances which can solve the instance in TSPLIB with 85,900 cities [5]. An efficient implementation of this approach is written by David Applegate, Robert Bixby, Vasek Chvátal and William Cook in ANSI C, which is called Concorde TSP solver [4].

Although exact algorithms guarantee the optimality of the solution found for TSP, they all have the weakness that the computational complexity grows exponentially with the problem size.

2.4.1.2 Heuristics and Approximation Algorithms for TSP

Except for exact algorithms, there are various approximation algorithms that provide good solutions to the TSP in reasonable computational time.

The approximation ratio of an algorithm A for a given instance I is defined as

$$\alpha_A(I) = A(I)/OPT(I),$$

where $A(I)$ refers to the fitness value of the solution found by algorithm A and $OPT(I)$ is the fitness value of the optimal solution to instance I . In the TSP, $A(I)$ and $OPT(I)$ refer to the total length of the tour found by the certain algorithm and the shortest tour respectively. An algorithm is called an r -approximation algorithm when $\alpha_A(I) \leq r$ holds for all instances I . The TSP discussed in this section is metric TSP.

The most straightforward heuristic algorithm for TSP is based on the nearest neighbour, which is a greedy algorithm. The algorithm starts with a city and selects one of its nearest neighbours as the next stop. This process repeats until all cities are visited. The runtime bound for this approach is $O(n^2)$ [84]. However, there exist many worst cases where this approach is not able to provide a good solution [65].

Another greedy approach constructs a tour gradually by selecting the edge with the shortest length. The algorithm terminates when it finds a Hamiltonian cycle. The time complexity is bounded above by $O(n^2 \log n)$ [133].

There is a 2-approximation algorithm based on *Minimum Spanning Tree* (MST), which solves metric TSP in polynomial time. As the first step, the algorithm constructs an MST T of the graph G representing the TSP instance and doubles the edges of T to form a new graph D . Then it finds an Eulerian tour E in D . The Hamiltonian cycle obtained after skipping all visited nodes of E is the result of the algorithm. This approach has a time complexity of $O(n^2 \log n)$.

The 3/2-approximation algorithm proposed by Christofides [21] is based on both the MST and minimum weight perfect matching. In this algorithm, after obtaining a MST T of the graph, a minimum weight perfect matching M is found for the odd-degree nodes of T . The algorithm combines all edges of M and T to form a graph C . Then an Euler cycle E is found in C . In the last step, the Hamiltonian cycle in S is produced by omitting all visited nodes in C . The time complexity of this approach is $O(n^3)$.

In [7, 6], Arora proposes a polynomial time approximation scheme (PTAS) for Euclidean TSP to find a solution with approximation ratio at most $(1 + 1/c)$ for any $c > 1$.

The 2-OPT algorithm is one of the local search algorithms for TSP. In this algorithm, a 2-OPT move refers to removing two edges from the tour and reconnecting the tour by swapping these two pairs of cities. Only if the 2-OPT move results in a shorter tour, it is accepted. The algorithm checks whether there is any 2-OPT move that can improve the current solution and terminates when no improvement is available and the shortest tour found so far is called *2-optimal*. The 3-OPT algorithm works in a similar way, while instead of swapping two edges in the tour, three edges are removed and the resulting unconnected nodes are reconnected to build a valid tour. The solution is called *3-optimal*. If a tour is 3-optimal, it is 2-optimal as well [73]. This approach is a good choice for large TSP instances.

As an extension of 2-OPT and 3-OPT algorithms, the Lin-Kernighan algorithm can be seen as a variable k -OPT algorithm. The suitable value of k is decided in each iteration. The average time complexity of the Lin-Kernighan algorithm is bounded by $O(n^{2.2})$ [73] which is slower than a classical 2-OPT algorithm. However, the results from the Lin-Kernighan algorithm are usually better than those from 2-OPT.

Besides the algorithms introduced above, there are many other algorithms using local search heuristics to solve the TSP, such as genetic algorithms [84], tabu search [46, 105], simulated annealing [105] and ant colony optimization [39].

2.4.2 Minimum Vertex Cover Problem

The *Minimum Vertex Cover* (MVC) problem is another well-known NP-hard combinatorial optimization problem with importance in both theory and application area. The MVC problem has many real-world applications, such as scheduling [10], VLSI design [62], industrial machine assignment [135] and sensor networks [140].

Given an undirected graph $G = (V, E)$, where V and E denote the set of vertices and edges respectively, a vertex cover is a subset $C \subset V$ such that for each edge $e \in E$ at least one of its endpoints is in C . The goal of the MVC problem is to find the smallest vertex cover in graph G and the problem has been proved to be NP-hard [126]. Therefore, there does not exist any algorithm that can optimize all MVC instances in polynomial time assuming $P \neq NP$. A set of vertices forms a vertex cover if and only if its complement is an independent set of the graph. The decision version of the vertex cover problem is to decide whether there exists a vertex cover of size at most k for a certain graph, which is also NP-complete.

The MVC problem is closely related two other well-known NP-hard combinatorial problems, which are *Maximum Independent Set* problem and *Maximum Clique* problem. MVC corresponds to the complement of maximum independent vertex set. The algorithms for the MVC problem can be directly used to solve the Maximum Clique problem which has a wide range of applications in computer vision, bioinformatics and computational chemistry [129].

There are some special graph types which are common subjects in the analysis of MVC problem. The bipartite graph refers to a graph whose vertex set can be divided into two disjoint sets U and V such that the edges in the edge set are all connecting these two sets. There is no odd-length cycle in a bipartite graph. In a complete bipartite graph, every vertex in set U is connected to every vertex in set V .

The tree graph is an undirected acyclic graph where any two vertices are connected by exactly one edge. There is no cycle in a tree but a cycle is formed if any edge is added to the graph. Every tree graph is a bipartite graph. There exists greedy algorithms that can solve the MVC problem in tree graphs through deep first search traversal with polynomial computational time.

Another special case is path where all vertices can be seen as connected by a single line. The path is one kind of tree graph. Except for the two terminal vertices which have degree 1, the other vertices are of degree 2. Paths are often included as subgraphs in other graphs.

Since the decision variant of the vertex cover problem is proved to be NP-complete, there may not exist efficient exact algorithm for the MVC problem. Researchers found that the decision problem can only be answered in time $O(1.2738^k + kn)$ [19].

The greedy algorithm for the MVC problem selects a node of the highest degree and the edges adjacent to this node are removed from the graph. The algorithm terminates when there is no edge in the graph. The algorithm performs well in solving some certain problems but in the worst case the greedy algorithm can only provide a $\Omega(\log n)$ solution.

The greedy 2-approximation algorithm which enumerating all edges can be improved by applying the linear programming formulation. In the Integer Linear Program (ILP) for MVC, a variable x_i is assigned to each node i . $x_i = 1$ if the node is selected and $x_i = 0$ if the node is not included in the set. The constraint is set to guarantee the full coverage of the edge set. The linear program relaxation can be solved using other algorithms for Linear Programming. This approach results in 2-approximation algorithms as well.

The heuristic algorithms for MVC problem are mainly stochastic local search algorithms. Although the heuristic algorithms cannot guarantee the optimality of the solutions, they can deal with large MVC instances by producing at least nearly optimal solutions in reasonable time. Since the MVC problem from real world applications are usually large in size and hard to solve, the studies into heuristic algorithms attract more and more attention. Popular local search algorithms designed for the MVC problem include PLS (Phased Local Search for the maximum clique problem) [129], NuMVC [16], TwMVC (Two weighting local search for MVC) [15] and COVER (Cover Edges Randomly) [135]. These approaches are usually evaluated on standard benchmarks and (in more recent years on) massive real world graphs.

2.5 Diversity in Evolutionary Algorithm

Different from the algorithms providing a single solution, population-based algorithms, such as genetic algorithms and ant colony algorithms, maintain a set of different solutions to the given problem called population. The diversity of the solutions in the population lies at the heart of most population-based algorithms, especially EAs. Most EAs incorporate certain diversity mechanisms which ensure that the population consists of a diverse set of individuals [17, 157]. A diverse population is one of the main factors to ensure that the solution space is explored adequately. The diversity in a population-based algorithm may point to different factors of the population. In EAs, the diversity in the objective space measures the variety of individuals in their values of evaluation functions. The decision space diversity describes the difference of individuals in the perspective of decision making, which includes the variety in different features of the individuals.

2.5.1 Diversity in Objective Space

EAs exploit information from previous generations to direct the search into the region of better performance [59]. In the early stage of the search, a diverse set of candidate solutions provides potential for the algorithms to explore the whole search space.

One of the problems facing all population-based metaheuristics including MOEAs is premature convergence to locally optimal solutions [99]. Diversity and the selective pressure are two factors that need to be considered as a trade-off. A good diversity maintenance mechanism is of great importance in preventing EAs from premature convergence. Hence,

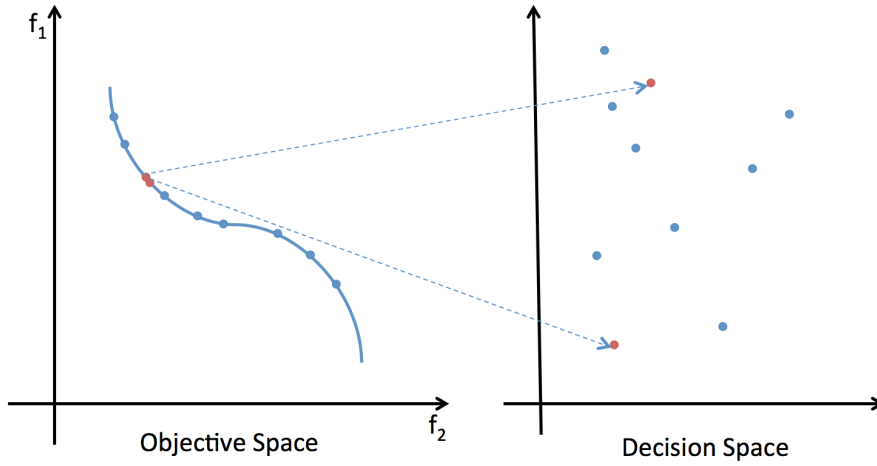


FIGURE 2.2: An example for a case that two individuals that are next to each other in objective space but are mapped to two points that are far away in decision space.

there have been many different mechanisms used to maintain the objective space diversity to balance the exploration and exploitation, e.g. niching, crowding [33] and sharing [60].

2.5.2 Diversity in Decision Space

Some researchers have pointed out that the diverse solution set in the decision space, which provides different solutions with good quality, is of great interest for decision makers [34, 139, 145]. A set of solutions that has a good coverage in the objective space does not guarantee a good coverage over the decision set since the adjacent points on the Pareto front may be mapped to points which are far away in decision space [145] as shown with an example in Figure 2.2. In this thesis, we focus on the discussion of diversity in the decision space which is important in both single-objective optimization and multi-objective optimization.

The history of integrating decision space diversity into the optimization process can be traced back to 1994 as the idea of fitness sharing in the NSGA for MOP [149]. Later, the idea of including genetic diversity as an objective is proposed in [152]. In the design of Omni-optimizer, the idea of NSGA was extended [34]. Recently researchers look into integrating the decision space diversity into the hypervolume-based multi-objective search [154] and single-objective optimization [156].

The solutions with minor difference in objective function may be various in structure or other perspectives. Therefore it is worthwhile to look into the decision-space diversity maximization in single-objective problems. Evolutionary approaches to obtain search space diversity under the condition that all solutions are of good quality have recently been proposed in the context of single- and multi-objective optimization [156, 154, 155]. In contrast to the standard approach of using diversity to obtain a single solution of high quality, the goal of these studies is to achieve a diversity set of solutions under the constraint that all solutions are of good quality.

While the target of most population-based EAs in solving single-objective optimization problem is to find a single high quality solution, the use of a population offers the opportunity to produce a diverse set of solutions which are all of good quality. Although there is usually just one best solution existing for a single-objective optimization problem, the decision maker may be interested in a set of acceptable-quality solutions which are of different structures. Typically, the aim of most single-objective problems is to find a solution which maximize or minimize a certain function, while there may be other factors that affects the decision of decision makers. In this way, a decision maker gets presented several good solutions that he/she can choose from in contrast to just a single best solution. The benefit in obtaining such a set of solutions lies in the variety of solutions and the intuitive knowledge of decision makers such as engineers to incorporate their personal knowledge when picking a solution for the final implementation. This is especially important as practitioners in the areas of engineering and manufacturing may have a specific preference for certain solutions even if they have similar quality according to the fitness function used to evaluate the quality of a given solution.

Although diversity mechanism is the key to the working behaviour of evolutionary multi-objective algorithms, maintaining decision space diversity in MOEAs is difficult since the coverage over Pareto front should be guaranteed at the same time. By maximizing the decision space diversity, a set of Pareto optimal solutions that differ to each other according to the underlying search space is presented to the decision makers. Such a set of solutions can be very valuable to decision makers since they can judge the solutions in an intuitive way without necessity to compare the quality of solutions using the objective functions.

2.5.3 Diversity Measurement

Before defining the population diversity, how to measure the difference between individuals should be decided first. There are many ways to measure the difference between different individuals and the choice of difference measurement depends on the individual representation and the main aim of the optimization process. The structural distance is one of the population difference measurement, such as Hamming distance and Euclidean distance. As an example, Hamming distance can be seen as a proper difference measurement for binary strings.

The diversity measurement is based on the definition of difference. According to [154, 155], a population diversity measurement should have the following properties:

1. Twinning: Duplicate solutions in a population should not change the diversity.
2. Monotonicity in Varieties: Adding a new solution which is not in a population should increase the set diversity.
3. Monotonicity in Distance: $D(P') \geq D(P)$ with $|P| = |P'|$ holds, if all pairs of P' are at least as dissimilar as all pairs of P (according to some given distance function).

The population diversity measurement should be designed to fulfil these requirements and be beneficial to the optimization problem. In some cases, the contribution of a solution to the population diversity is defined instead of directly defining the diversity measurement. The contribution to the population diversity is helpful in the selection process based on the decision space diversity of EAs.

2.6 Conclusion

In this chapter, we give a brief background introduction about the combinatorial optimization and heuristic search. Combinatorial optimization problems attract attention of researchers and practitioners since there are many real world complex problems that can be represented as combinatorial optimization problems. There have been many studies into the combinatorial optimization problems in the area of algorithm design, problem characterization and theoretical analysis [126].

Heuristic search provides a way to solve various combinatorial optimization problems successfully. Although heuristic algorithms usually do not guarantee the optimal solutions to be found, they obtain solutions of good quality in reasonable time. Heuristic algorithms, such as local search, evolutionary algorithm and ant colony optimization, have shown good performance under the investigation from both theoretical and practical aspects.

Although the world is full of suffering, it is also full of the overcoming of it.

Helen Keller

CHAPTER 3

ALGORITHM ANALYSIS AND PROBLEM COMPLEXITY IN EVOLUTIONARY ALGORITHMS

3.1 Introduction

There has been rapid development of evolutionary algorithms in algorithm design and application, however, the theoretical analysis of EAs is still far behind the practical experiments. The theoretical analysis of EAs is often surprisingly difficult [119]. The main reason may lie in the fact that EAs are randomized search heuristics inspired by the natural behaviours of evolution and are designed to perform the search in wide search space guided by random decisions without analyzing behaviours. Moreover, EAs are often considered as robust problem-independent search heuristics with good performance on a large variety of problems, which makes the analysis of EAs much harder than the analysis of problem-specific algorithms [159]. Nevertheless, the theoretical analysis provides understanding of the characteristics and behaviours of EAs, which is beneficial for the design and application of EAs. It has been pointed out that improving the theoretical understanding and performance prediction is one of the most challenging problems facing the community of optimization and algorithms [126].

In the past few decades, great improvements have been achieved by the computational complexity analysis of EAs in the theoretical understanding of the algorithms. However, until

the early 1990s, the research into theory on EAs mainly focus on investigation of the convergence of EAs or analysis of EAs' behaviour in one single generation [119]. The first runtime analysis on EA was proposed in 1992 by Mühlenbein [113]. After that, a lot of analyses have been conducted on EAs and many useful methods were introduced.

This chapter focuses on a general overview about the algorithm analysis methods in evolutionary algorithms. In Section 3.2, some classical computational complexity analysis methods are introduced. Section 3.3 and 3.4 provide a brief discussion about two popular computational analysis methods in different aspects, which are parameterized analysis and feature-based analysis.

3.2 Classical Computational Complexity Analysis

As a class of randomized algorithms, there are a lot of strong methods can be used in the analysis of EAs [8, 80]. In the early stage of runtime analysis of EAs, the research mainly focuses on the analysis of artificial pseudo-boolean functions [42, 69, 138, 159, 162]. In these previous studies, the simple pseudo-boolean functions are examined to provide efficiency analysis together with introduction of new analysis techniques and the design of a general Markov chain framework for the runtime analysis on EAs [71]. Later on the work extended to more generalized problem types, such as linear functions [42], quadratic polynomials [160] and some combinatorial optimization problems [56, 118, 121]. By looking into the search behaviours of these algorithms, these studies show that the general-purpose algorithms can solve or provide good approximations for classical problems.

In this section, some useful tools for computational complexity analysis are introduced, including fitness-based partitions, some deviation bounds and drift analysis.

3.2.1 Fitness-based Partitions

Fitness-based partition is a simple method proposed by Wegener [159] as one of the early methods for runtime analysis of EAs. It has been successfully applied to the analysis of many problems. The fitness-based partition is often used to derive an upper bound on the expected optimization time for a certain optimization problem.

Assume the problem under investigation is in a search space S and the objective function to be maximized is $f : S \rightarrow \mathbb{R}$. When A_i and A_j are two subsets of S , define $A_i <_f A_j$ as $f(a) < f(b)$ holds for all $a \in A_i$ and all $b \in A_j$. Divide S into disjoint sets $A_0, A_1, \dots, A_m$ such that $A_0 <_f A_1 <_f \dots <_f A_m$ holds and A_m consists of all optimal solutions. This implies that the fitness values of the solutions in each partition increase with increasing the index. The collection of set $A_0, \dots, A_m$ is called an *f-based partition*.

Let $p(x)$ represent the probability that from a solution $x \in A_i$, in the next step a solution $x' \in A_{i+1} \cup \dots \cup A_m$ is generated and $p_i = \min_{a \in A_i} p(x)$ represents the smallest probability that a solution from a partition with higher fitness value is generated.

Lemma 3.1. *The expected optimization time of a stochastic search algorithm that at each time step works with a population of size 1 and produces a new solution from the current solution is upper bounded by $\sum_{i=1}^{m-1} (1/p_i)$.*

It can be proved easily with probability distribution knowledge and the proof can be found in [119]. With a proper partitioning of the search space which guarantee a high probability of leaving the current partition and the above Lemma, the upper bound of many optimization problems with $(1 + 1)$ -EA is reached, such as OneMinMax, LeadingOnes and linear functions [80].

3.2.2 Deviation Inequalities

The application of large deviation inequalities has great contribution to the analysis of randomized algorithms. In the case of EAs, the deviation bounds can be used to prove the probabilities that the actual running time can deviate from the expected optimization time. Some of the commonly used deviation bounds are Markov's inequality, Chebyshev inequality and Chernoff bounds. In the following paragraphs, we give a brief introduction about these bounds without proof. The in-detail proof can be found in the textbook by Motwani and Raghavan [112].

Markov's Inequality: Let X be a non-negative random variable. Then for all $k \in \mathbb{R}_{>0}$,

$$\text{Prob}(X \geq k \cdot E(X)) \leq 1/k.$$

k is not restricted to integer. Since the expectation of X

$$E[X] \geq \text{Prob}[X \geq t] \cdot t + \text{Prob}[X < t] \cdot 0 = \text{Prob}[X \geq t] \cdot t,$$

it is proved that

$$\text{Prob}[X \geq t] \leq E[X]/t.$$

Chebyshev's Inequality: Let X be a random variable with mean μ and standard deviation σ . Then for any $k \in \mathbb{R}_{>0}$,

$$\text{Prob}[|X - \mu| > k \cdot \sigma] \leq 1/k^2.$$

Chernoff Bounds: Let $X_1, X_2, \dots, X_n$ be independent Poisson trials such that for $1 \leq i \leq n$, $\text{Prob}[X_i = 1] = p_i$, where $0 \leq p_i \leq 1$. Let $X = \sum_{i=1}^n X_i$ and $\mu = E(X) = \sum_{i=1}^n p_i$. Then the following inequalities hold,

$$\text{Prob}[X \geq (1 + \delta) \cdot \mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu, \delta > 0;$$

$$\text{Prob}[X \geq (1 + \delta) \cdot \mu] \leq e^{-\mu\delta^2/3}, 0 < \delta \leq 1;$$

$$\text{Prob}[X \leq (1 - \delta) \cdot \mu] \leq e^{-\mu\delta^2/2}, 0 < \delta \leq 1.$$

In the coupon collector's problem discussed in [112], given n different coupons, at each step one of the n coupons is chosen uniformly at random. The target is to find the number of trials until each of the coupons has been chosen at least once.

Coupon Collector's Theorem: In the coupon collector's problem, at each trial one of the n different coupons is chosen uniformly at random. Let X represent the number of trials required to choose each coupon at least once. Then

$$E(X) = n \cdot H_n,$$

where H_n refers to the n th Harmonic number and for each constant $c \in \mathbb{R}$,

$$\lim_{n \rightarrow \infty} \text{Prob}[X \leq n \cdot (\ln n - c)] = e^{-e^c}.$$

There are also some elementary mathematical formulas which are helpful in runtime analysis. We will state in this section without proof as well.

Harmonic Sum: Let $H_n = \sum_{i=1}^n \frac{1}{i}$ represent the n th Harmonic sum. Then for any $n \in \mathbb{N}$

$$H_n = \ln n + \Theta(1).$$

Stirling's Formula: Let $n \in \mathbb{N}$, then

$$\sqrt{2\pi n} \cdot n^n e^{-n} < n! < \sqrt{3\pi n} \cdot n^n e^{-n}.$$

Binomial Coefficients: Let $n \geq k \geq 0$. The binomial coefficients are defined as

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}.$$

Then the following inequality holds,

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \frac{n^k}{k!} \leq \left(\frac{ne}{k}\right)^k.$$

Inequalities regarding e :

$$e^x \geq 1 + x, x \in \mathbb{R};$$

$$e^{-x} \leq 1 - x/2, 0 \leq x \leq 1;$$

$$e^x \leq \frac{1}{1-x}, x < 1;$$

$$\left(1 - \frac{1}{n}\right)^n \leq \frac{1}{e} \leq \left(1 - \frac{1}{n}\right)^{n-1}, n \in \mathbb{N}.$$

These formulas are often used together with the deviation inequalities stated above and for the introduction and proof in details the readers are referred to the textbook by Feller [45].

3.2.3 Drift Analysis

Another popular algorithm analysis method of EAs in recent years is drift analysis which is introduced by He and Yao in 2001 [69, 68] based on the results of Hajek [66]. Drift analysis is a powerful tool in analyzing the optimization behaviour of a randomized search heuristic. The following studies introduce the simplified drift [123], multiplicative drift [38], population drift [97] and variable drift [81] to deal with different situations.

In drift analysis, instead of examining the improvement over the objective function, an auxiliary potential function is used and its behaviour is tracked to provide an idea of the optimization process in search space. The potential function maps each search point to a non-negative real value, where the value of 0 indicates the optimal search point.

Let $X_0, X_1, \dots, X_t$ represent some stochastic processes in some general state space $\mathcal{X}$. The random variable corresponding to each process is denoted by $\{X^t\}$ over $S \cup \{0\}$, where $S \subset \mathbb{R}$. The expected change in one step of the random process is called *drift*. Assume random variable T refers to the first point in time $t \in \mathbb{N}$ with $X^t = 0$.

Theorem 3.1. (*Addictive Drift*) Suppose that there is a real number $\Delta > 0$ that fulfils

$$E[X^t - X^{t+1} | T > t] \geq \Delta.$$

Then the expected optimization time

$$E[T | X^0] \leq X^0 / \Delta.$$

A suitable potential function is essential for the application of additive drift analysis. For example, for the sequence of random search points generated by a homogeneous absorbing Markov chain on S , the drift $\Delta := E[X^t - X^{t+1} | T > t]$ satisfies the requirement with equality.

The multiplicative drift theorem is not as strong as the classical additive drift, however, it allows us to use natural potential functions [38]. Let $s_{\min}$ be the minimum value of the set S .

Theorem 3.2. (*Multiplicative Drift*) Suppose that there is a real number $\Delta > 0$ that for all $s \in S$ with $\text{Prob}[X^t = s] > 0$ it fulfils

$$E[X^t - X^{t+1} | X^t = s] \geq \Delta \cdot s.$$

Then for all $s_0 \in S$ with $\text{Prob}[X^0 = s_0] > 0$, the expected optimization time

$$E[T | X^0 = s_0] \leq \frac{1 + \ln(s_0/s_{\min})}{\Delta}.$$

For the analysis of combinatorial optimization process, the distance between objective value of the current solution and the optimal value is often a proper choice of potential function and it fulfils the requirement of multiplicative drift.

Many recent improvements in the area of complexity analysis of EAs are based on drift analysis. Some applications of drift theorems on example problems can be found in Chapter 5.3-4 of the textbook of Jansen [80].

3.3 Parameterized Analysis

Parameterized runtime analysis aims at obtaining knowledge about how the problem structure of problem influences the algorithmic runtime. In parameterized analysis, the source of exponential complexity in NP-hard problems is often isolated from the other parts to make the examination of problem hardness based on problem structure possible [41]. The idea of parameterized runtime analysis is to partition the problem based on parameters related to the structure of instances and evaluate the problem complexity on the basis of the separation. Parameterized analysis provides a measurement of the problem complexity with multiple input parameters.

The parameterized analysis of heuristic search methods has gained a lot of attention during the last few years [150, 94, 28, 95, 114]. The analysis is guided by the structural properties of the problem instances. It provides a mechanism for understanding how and why heuristic methods work for prominent combinatorial optimization problems.

3.3.1 Some Basic Definitions

A *parameterized* problem refers to a language $L \subseteq \Sigma^* \times \Sigma^*$. If (x, k) is in a parameterized language L , k is called the *parameter*. The parameter k is often a positive integer, but there exist situations where k is a graph or an algebraic structure [41].

Definition 3.1. A parameterized problem L is *fixed-parameter tractable (FPT)* if it can be determined in $f(k) \cdot |x|^{O(1)}$ time whether $(x, k) \in L$, where f denotes a computable function which only depends on the parameter k . The algorithm involved in the decision is called a *fixed-parameter tractable algorithm*.

In the parameterized runtime analysis of EAs, the expected number of generations for the algorithm to decide on a parameterized decision problem is the main goal to reach. A randomized algorithm which has expected optimization time $E[T] \leq f(k) \cdot n^{O(1)}$ is defined as a randomized FPT algorithm for the corresponding parameter k .

The big O^* notation which is common in parameterized analysis is proposed to simplify the expression of parameterized complexity by omitting all terms of lower order.

Definition 3.2. *If a parameterized algorithm has expected running time $f(k) \cdot |x|^c$, the algorithm computation time can be represented by $O^*(f(k))$ by ignoring the polynomial part and focusing on the exponential part.*

3.3.2 Bounded Search Tree

One popular paradigm to design parameterized algorithms is the *bounded search tree algorithm* which searches for a good solution by branching according to different rules that may be applied to solve the underlying problem. The search space is often represented as a search tree with the size bounded by a function of parameters. Constructing the search tree is considered as the first step of the method of bounded search trees. Then some relatively efficient algorithm is executed on each branch of the tree. The worst case for complexity of such algorithms is when a complete exploration of the search space is necessary. In practical implementations, one of the key strategies is to reduce branchings whenever possible. It is often effective to integrate the bounded search tree approach to other mechanisms, such as kernelization [41].

An FPT algorithm for the Minimum Vertex Cover problem is simple to construct based on the bounded search tree. For an arbitrary edge in the graph, one of its adjacent vertices should be included in the solution set in order for the edge to be covered. The parameter k here is the size of vertex cover for the graph. Therefore the idea is to branch according to these two alternatives.

Theorem 3.3. *The decision variant of Minimum Vertex Cover problem is solvable in time $O(2^k \cdot |V|)$.*

The hidden constant in the expected runtime is independent of parameter k and the number of vertices $|V|$.

Theorem 3.3 can be proved by analyzing the binary tree of height k constructed following certain rules and the detailed proof can be found in [108, 40].

Some improvements can be achieved by shrinking the search tree. If in the graph G there is no vertex of degree 3 or more, then the graph is formed only by paths, cycles and isolated vertices, which makes the MVC problem easy to be solved. For such graphs with more than $2k$ edges, the size of vertex cover cannot be less than k . Otherwise, the graph have many vertices of degree 3 or more. Choose one of these vertices and the possible move is either selecting the certain vertex or selecting all its neighbours. Selecting the certain vertex results in covering the rest of graph with $(k - 1)$ vertices, while selecting the w neighbours means that the rest of the graph should be covered with $(k - w)$ nodes. Then the size of vertex cover can be represented in a recursive way, which leads to the following improved result.

Theorem 3.4. *The decision variant of Minimum Vertex Cover problem is solvable in time $O(1.466^k \cdot |V|)$.*

This solution is feasible for $k \leq 70$. The state-of-art algorithm in [18, 20] provides a solution in time $O(1.286^k + k \cdot |V|)$ which also involves other techniques to improve the performance.

3.3.3 Kernelization

Another key concept in designing FPT algorithm is *kernelization*. Given an input (I, k) , the main idea of kernelization is to eliminate the problem instance I to an "equivalent" instance I' for which the size is bounded by a function depending only on parameter k . The input (I, k) with answer YES can be transformed into (I', k') iff the answer to (I', k') is YES.

Definition 3.3. Let L be a parameterized problem consisting of (I, k) , where I and k denote the problem instance and parameter respectively. Then reduction to a problem kernel means replacing (I, k) with a reduced instance (I', k') which is called problem kernel such that the following property fulfils:

1. $k' \leq k$ and $|I'| \leq g(k)$ for some function g which only depends on k .
2. $(I, k) \in L$ iff $(I', k') \in L$
3. The reduction from (I, k) to (I', k') has to be solvable in polynomial time.

Kernelization is a technique of pre-processing and small kernels yield quick algorithms.

3.4 Feature-based Analysis

Heuristic search methods such as local search, simulated annealing and evolutionary algorithms have shown good performance in solving various combinatorial optimization problems. As stated in the no-free-lunch theorems [163], we should not expect there exists a single algorithm that can outperform all other algorithms in all instances. Therefore, understanding the conditions under which these algorithms give good performance is one of the essential preconditions for automatic algorithm selection, configuration and algorithm design [165, 102]. Many studies have been conducted from the perspective of theory and application in order to obtain comprehensive understanding of these conditions [29]. The actual behaviour of algorithms is hardly captured insightfully only by the research into the worst case or benchmark datasets [77]. It is often hard to predict the performance of an algorithm on a certain instance of a combinatorial optimization problem without running the algorithm on it [48, 85]. Hence, the feature-based analysis is proposed to investigate the behaviour of algorithms on instances before actual execution of the algorithm.

The feature-based analysis of heuristic search algorithms has become an important topic in understanding such type of algorithms [109, 146]. This approach characterizes algorithms and their performance on a given problem based on feature values of the problem instances. Thereby, it provides an important tool for bridging the gap between pure experimental investigations and mathematical methods for analyzing the performance of search algorithms [119, 91, 44].

In the artificial intelligence area, a good measure of the key characteristics of optimization problem leads to the successful regression model for prediction of algorithm selection on new problem instances, which is named as algorithm portfolio [61, 100]. In the operational research communities, the research into the feature-based analysis on understanding problem difficulty provides methodology to determine a good selection of features [148, 74, 146].

3.4.1 Features Selection for Characterizing Problem Instances

In 1976, Rice proposed the problem of algorithm selection in [134] which focuses on obtaining a mapping from the feature space to algorithm performance space and emphasizes the importance of appropriate features for characterizing the hardness of problem instances. After that, although the importance of feature-based analysis is highlighted by many researchers, not enough attention has been paid to construct suitable features for characterizing problem instances as a preliminary step for algorithm selection and performance modelling [147].

There are various features for each combinatorial optimization problem, however, most of these features are not useful in the analysis of algorithm performance and some of the features are hard to be measured [147]. A proper selection of candidate features based on the research problem type makes great contribution in the analysis process. Throughout the whole thesis, our main focus of feature-based analysis is the hardness-revealing features for combinatorial optimization problems.

Features for exposing the hardness of instances can be divided into two main types based their evaluation objects, which are problem-independent features and problem-specific features [147]. One popular approach to characterize the difficulty of an optimization problem without problem-specific knowledge is fitness landscape analysis [132, 144]. The fitness landscape provides a holistic overview of the search space. The problem-independent features are always combined with some problem-specific features to provide insight of problem difficulty. In this thesis, we focus on the discussion of problem-specific feature analysis.

As an example, we examine the Euclidean traveling salesman problem which is one of the combinatorial optimization problems discussed in Chapter 2. There have been many studies investigating the problem-specific features for TSP in recent years [146, 87, 109, 147, 115]. Most of the features of interests are related to the structure of the problem instances. Some of the features discussed in the previous studies can be classified into the following groups.

Distance Features: The statistical figures of the edge cost distribution are considered as distance features. The minimum, maximum, mean and median value of the edge costs and other summary statistics fall in this type of features.

Distribution Features: The summary statistics describe the distribution of cities of the TSP instance are the distribution features. The locations of cities are summarized to a single value that provide some insights into the aggregation and the overall structure.

Mode Features: The number of modes and other related statistics of the edge cost distribution are classified as the mode features. However, these features have limitation in randomly generated instances.

Cluster Features: As recommended in [146, 109], GDBSCAN [142] can be used to find clusters in TSP instances with different reachability distances. The statistics gathered are treated as cluster features.

Nearest Neighbour Distance Features: The statistics related to the normalized nearest neighbour distance among each pair of cities in the TSP instance describe the uniformity of the instance.

Centroid Features: The centroid features include all values which are relevant to the distance between other nodes and the instance centroid.

Minimum Spanning Tree Features: The minimum spanning tree (MST) constructed from the TSP instance provides another type of features. The statistics related to the depth and distances of the MST are common choices of features.

Angle Features: This type of features describes the angles between a city and two of its nearest neighbours. Same as other types, the statistics such as maximum, minimum, standard deviation and other numerical values are included.

Convex Hull Features: As the area of the convex hull of the TSP instance can be seen as a reflection of how disperse the cities are in the plane, this type of features is also popular.

For a more detailed introduction of all popular structural features for TSP, we refer the readers to [109] for more information.

3.4.2 Feature-based Analysis for Problem Hardness

The problem hardness analysis is often based on examination of large scale of datasets for the algorithm behaviours in order to identify the relationship between the features and the contribution of each feature to the problem hardness for a certain algorithm.

How to evaluate the hardness of certain instance is the first problem facing the researchers. The performance of a certain algorithm on an instance can be measured based on the quality of the solution or the running time for the obtained solution to reach certain quality. For approximation algorithms in solving TSP, the quality of the solution can be measured by the approximation behaviour which is often evaluated by the approximation ratio of the tour length. This is also an option for exact algorithm with time constraint. The running time for exact algorithm to reach the global optimum solution or exact and approximation algorithms to obtain solution satisfying certain requirement can also be regarded as relative hardness measurement.

The second problem is how to generate the instance set to work on. Current methods for the feature-based analyses are based on constructing hard and easy instances for an investigated search heuristic and a given optimization problem by evolving instances using an evolutionary algorithm [109, 117, 116]. This evolutionary algorithm constructs problem instances where the examined algorithm either shows a bad (good) approximation behaviour and/or requires a large (small) computational effort to come up with good or optimal solutions. Although the evolutionary algorithm for constructing such instances is usually run several times to obtain a large set of hard or easy instances, the question arises whether the instances obtained give a good characterization of problem difficulty in terms of features.

Another problem follows the generation of instance set is the diversity of instance set. In order to have a more comprehensive overview over the search space which can be used for the prediction of new instances, the dataset to be studied should be diverse enough to provide useful insights. With the evolutionary process of generating instances, different instances can be produced for examination. However, it is not guaranteed that these instances are all different and the diversity is not measured. In Chapter 8 we discuss a new approach of constructing a diversity set of hard or easy instances.

3.5 Conclusion

The theoretical understanding of evolutionary algorithms is very important for algorithm design, algorithm selection and application. However, analyzing the behaviour of EAs is often surprisingly difficult, even for very simple EAs on some simple artificial functions. In this chapter, we give a brief discussion about some useful methods of algorithm analysis which lays out the foundation for analyzing EAs. These methods are easy to understand and applicable in many different situations. In the following chapters, these methods are used for conducting runtime analyses on specific problems.

Except for the classical complexity analysis methods, there are many novel analyzing methods which attract more and more attention from both the theoretical and practical perspectives, which includes parameter complexity analysis and feature-based analysis. Both approaches have been successfully applied to the analysis of heuristic search methods [150, 94, 28, 114, 109]. They are all often guided by the structural properties of the problem instances under investigation and provide some insights into the performance of algorithms on different instances. In later chapters, we will discuss some complexity analyses based on the methods discussed in this chapter.

An experiment is a question which science poses to Nature, and a measurement is the recording of Nature's answer.

Max Planck

CHAPTER 4

HEURISTIC ALGORITHMS FOR MINIMUM VERTEX COVER PROBLEM

4.1 Introduction

The Minimum Vertex Cover (MVC) problem can be regarded as one of the prominent NP-hard combinatorial optimization problems with many real-world applications [55]. It has been proved that it is NP-hard to approximate the MVC problem with approximation ratio smaller than 1.3606 [36]. There are many exact algorithms and heuristic algorithms designed for solving MVC. However, the state-of-art algorithms can only approximate MVC by a factor of $2 - o(1)$ [67, 88]. In Chapter 2, we include a brief introduction of the MVC problem about its formulation and algorithms.

Local search algorithms belong to the most successful approaches for many combinatorial optimization problems [1, 78]. Heuristic algorithms, including local search algorithms, may not be able to guarantee the optimality of the solutions, however, they can find nearly optimal solutions within reasonable computational time. Therefore, it is sufficient to apply the local search approach in solving hard and large MVC instances. There have been many local search algorithms proposed by researchers to solve the MVC problem and some of them show good performance [128, 16, 15, 135].

The parameterized analysis of heuristic search methods as introduced in Chapter 3, has gained a lot of attention during the last few years [150, 94, 28, 95, 114]. It provides a mechanism for understanding how and why heuristic methods work for prominent combinatorial optimization problems. One popular paradigm to design parameterized algorithms is *bounded search tree algorithm* which searches for a good solution by branching according to different rules that may be applied to solve the underlying problem. For classical MVC problem, different branching algorithms are available to answer the question whether a given graph has a vertex cover of size at most k . We investigate two common strategies in this chapter to solve the problem.

This chapter is based a conference paper published in the conference PPSN 2016 [51] .

In this chapter, we first introduce two parameterized local search algorithms. Then in Section 4.3, we include an investigation into the comparison between different initialization approaches and experimental results to support our arguments. Section 4.4 focuses on the introduction of two local search approaches for the MVC problem and with some experimental results we compare the behaviours of the two algorithms. At last, we finalize this chapter by some remarks.

4.2 Background

The MVC problem is one of the best-known combinatorial optimization problems. Given an undirected graph $G = (V, E)$, the goal is to find a minimum set of vertices V' such that each edge has at least one end vertex in V' . A detailed introduction of MVC is included in Chapter 2. This problem has been studied extensively in the area of parameterized complexity. In fact, it is the archetypical problem in this area. Various kernelization approaches leading to fixed parameter algorithms of different runtime quality are known.

We make use of two branching approaches in this study which are both from the area of parameterized complexity [41]. Both have been introduced to determine whether a given graph $G = (V, E)$ contains a vertex cover of at most k nodes. The first approach builds on the fact that a vertex cover has to contain for each edge at least 1 node. It starts with an empty set, picks an edge $e = \{u, v\}$ currently not covered, and branches according to the two options of including u or v . This allows the user to answer the question of whether G contains a vertex cover of size at most k or not in time $O^*(2^k)$.

The second approach makes more sophisticated decisions according to the degree of a node with respect to the uncovered edges. Considering a degree 1 node, it's always safe to take its neighbor. In the case of dealing with a degree 2 node u , one has to choose either the two neighbors v and w of u or all neighbors (including u) of v and w . Finally, for a node u of degree at least 3, one has to choose u or all its neighbours. This approach makes it possible to answer the question of whether G contains a vertex cover of size at most k or not in time $O^*(\alpha^k)$, where $\alpha = 1.4656$.

Algorithm 4.1: Edge-based Branching Initialization Heuristic

```

1  $C := \emptyset$ ;
2 repeat
3   Let  $e = \{u, v\}$  be a random uncovered edge, i.e.,  $e \in G[C]$ ;
4   with probability  $1/2$  do
5      $C := C \cup \{u\}$ 
6   else
7      $C := C \cup \{v\}$ 
8 until  $C$  is a vertex cover of  $G$ ;
9 Return  $C$ ;

```

We build on these two fixed parameter algorithms for the decision version of the vertex cover problem and study how to turn them into randomized initialization strategies with provable guarantees on their probability of achieving a solution of certain quality. In addition, we explore how they can be turned into local search approaches and study the performance of these approaches on benchmark instances.

For describing our algorithms, the following notation is added. For each vertex cover $C \subseteq V$ of a graph $G = (V, E)$, we denote the subgraph of G consisting of the edges not covered by C and the corresponding non-isolated vertices by $G[C] := (V_C, E_C)$ with

$$E_C := E \setminus \{e \in E \mid e \cap C \neq \emptyset\} \text{ and}$$

$$V_C := \{v \in V \mid v \cap E_C \neq \emptyset\}.$$

Furthermore, we represent the degree of a node u in $G[C]$ and the set of neighbours of u in $G[C]$ by $\deg_{G[C]}(u)$ and $N_{G[C]}[u]$ respectively.

4.3 Initialization Strategies

Local search algorithms often start with some initial solution and refine it by small local search moves. A good initial solution provides a starting point with better fitness value and/or higher possibility to avoid getting stuck in local optima.

4.3.1 Different Initialization Approaches

Firstly, two basic randomized initialization strategies based on the branching approaches as described in the previous section are introduced. Both of these approaches start with an empty set of nodes and add vertices until a vertex cover has been obtained.

The edge-based initialization with branching outlined in Algorithm 4.1 randomly selects in each step an uncovered edge and adds one of its endpoints chosen uniformly at random to the vertex cover. The search step is iterated until a vertex cover is achieved.

Algorithm 4.2: Vertex-based Branching Initialization Heuristic

```

1  $C := \emptyset$ ;
2 repeat
3   if  $\text{mindeg}(G[C]) = 1$  then
4     Let  $u$  be a random node with  $\deg_{G[C]}(u) = 1$ ;
5      $C := C \cup N_{G[C]}[u]$ ; /* degree 1 rule */
6   else
7     switch depending on the variant of the algorithm do
8       case mindeg variant do
9         Let  $u$  be a random node with  $\deg_{G[C]}(u) = \text{mindeg}(G[C])$ ;
10      case maxdeg variant do
11        Let  $u$  be a random node with  $\deg_{G[C]}(u) = \text{maxdeg}(G[C])$ ;
12      case uniform variant do
13        Let  $u$  be a node chosen uniformly at random from  $G[C]$ ;
14      case degree-proportional variant do
15        Choose  $u$  with probability  $\deg_{G[C]}(u) / \sum_{v \in G[C]} \deg_{G[C]}(v)$ ;
16      if  $\deg_{G[C]}(u) = 2$  then
17        Let  $v, w \in V$  such that  $N_{G[C]}[u] = \{v, w\}$ ;
18        with probability  $\alpha^{-|N_{G[C]}[v] \cup N_{G[C]}[w]|}$  do
19           $C := C \cup N_{G[C]}[v] \cup N_{G[C]}[w]$ 
20        else
21           $C := C \cup N_{G[C]}[u]$ ; /* degree 2 rule */
22      else
23        with probability  $\alpha^{-\deg_{G[C]}(u)}$  do
24           $C := C \cup N_{G[C]}[u]$ 
25        else
26           $C := C \cup \{u\}$ ; /* degree  $\geq 3$  rule */
27 until  $C$  is a vertex cover of  $G$ ;
28 Return  $C$ ;

```

We now introduce an initialization heuristic based on more complex vertex-based branching rules. The vertex-based initialization given in Algorithm 4.2 first handles degree 1 nodes in the graph $G[C]$. The neighbour of the degree 1 node is selected. Please note that the degree of each vertex is calculated based on graph $G[C]$, not the original graph G .

If there is no degree 1 node in $G[C]$ then a node u in $G[C]$ is chosen based on the variant of the algorithm. With *mindeg*, *maxdeg* and *uniform* variant, the node to branch on is selected randomly from the vertices with minimum degree, maximum degree and all of the uncovered node set respectively. For *degree-proportional* variant, every uncovered node is given a probability based on its degree to be selected as node u . Then the degree rule for u is applied in a probabilistic way. To be more precise, if u is of degree 2 and v, w are its two neighbours in $G[C]$ then all neighbours of v and w are added with probability $\alpha^{-|N_{G[C]}[v] \cup N_{G[C]}[w]|}$, or v and w are added otherwise. Similarly, if u is of degree at least 3 in $G[C]$ then all neighbours of u in $G[C]$ are added with probability $\alpha^{-\deg_{G[C]}(u)}$, or u is added otherwise.

Algorithm 4.3: Greedy Initialization Heuristic

```

1  $C := \emptyset;$ 
2 repeat
3   Let  $u$  be a randomly selected uncovered vertex in  $G[C]$  with maximum degree, i.e.,
    $u \in G[C]$  with  $\deg_{G[C]}(u) = \maxdeg(G[C]);$ 
4    $C := C \cup \{u\}$ 
5 until  $C$  is a vertex cover of  $G$ ;
6 Return  $C$ ;
```

Algorithm 4.4: Node-based Initialization Heuristic

```

1  $C := V;$ 
2 for each node  $u$  in  $C$  do
3   if  $C \setminus \{u\}$  is a vertex cover of  $G$  then
4      $C := C \setminus \{u\}$ 
5 Return  $C$ ;
```

Another popular initialization process is the greedy way as shown in Algorithm 4.3. This approach starts with an empty vertex set and selects an uncovered vertex with the maximum degree in graph $G[C]$ randomly. This step is iterated and the initialization process terminates until the current vertex set is a vertex cover. It is similar to Algorithm 4.1 but behaves differently in practice.

Apart from the vertex-based branching heuristic, there is another vertex-based approach for initialization which starts with the whole vertex set V and refine the set in a greedy way. As the pseudo code in Algorithm 4.4 shows, this approach checks whether removing a certain vertex from the current solution set results in a vertex cover or not following an order decided in the beginning of the algorithm.

4.3.2 Theoretical Analysis

For the edge-based initialization process we provide the proof for a tradeoff between the size of the obtained vertex cover and the success probability.

Theorem 4.1. *For all r with $0 \leq r \leq OPT$, the edge-based initialization heuristic obtains a vertex cover of size at most $k := 2 \cdot OPT - r$ with probability at least $\binom{k}{OPT} \cdot 2^{-k}$.*

Proof. Let C^* be an optimal solution of value OPT . For each edge e at least one of its endpoints is contained in C^* . Hence, each step in the initialization process increases the number of nodes chosen from C^* by 1 with probability at least $1/2$. We call a step increasing the number of nodes already chosen from C^* a *success*. OPT successes are sufficient to obtain a vertex cover. The probability to have OPT successes during k steps is at least $\binom{k}{OPT} \cdot 2^{-k}$. $\square$

In some special cases, observe that for $r := 0$ (then $k = 2 \cdot OPT$), the edge-based initialization heuristic obtains a 2-approximation of the minimum vertex cover with probability at least

$\binom{2^{OPT}}{OPT} \cdot 2^{-2^{OPT}} = \Theta(1/\sqrt{OPT})$. On the other hand, for $r := OPT$ (and $k = OPT$), the edge-based initialization heuristic obtains a minimum vertex cover with probability at least 2^{-OPT} .

Then we provide a lower bound on the probability that the vertex-based initialization obtains an optimal solution.

Theorem 4.2. *The vertex-based initialization heuristic obtains a vertex cover of size OPT with probability at least α^{-OPT} , where $\alpha = 1.4656$.*

Proof. The vertex-based initialization heuristic carries out a randomized branching according to different rules. We distinguish the different cases regarding the degree of a node. For any graph, there is an optimal vertex cover that does not contain the node u if u is a degree one node. We investigate the degree 2 and 3 rules and show that each step i which requires selecting OPT_i nodes corresponding to an optimal solution occurs with probability at least α^{-OPT_i} .

For a degree 2 node, there is an optimal vertex cover that contains either the neighbors v and w of u or all the neighbors of v and w . Note that the degree 2 rule is only applied if there is no node of degree 1 in $G[C]$. This implies that both v and w have to be connected to a node different from u . The probability of selecting v and w is $1 - \alpha^{-|N_{G[C]}[v] \cup N_{G[C]}[w]|}$ which is at least α^{-2} if $|N_{G[C]}[v] \cup N_{G[C]}[w]| \geq 2$. If $|N_{G[C]}[v] \cup N_{G[C]}[w]| = 1$, then v and w are connected and we have a cycle of length 3 (which can be represented as $u-v-w-u$) for which selecting any subset of 2 nodes is optimal. Selecting u leads to an isolated edge $\{v, w\}$ for which the degree 1 rule selects a single vertex and therefore situations where $|N_{G[C]}[v] \cup N_{G[C]}[w]| = 1$ always lead to an optimal solution for the cycle of length 3.

Finally, if u is of degree at least 3, there is an optimal vertex cover which either contains u or all the neighbors of u . The probability of selection u is $1 - \alpha^{-\deg_{G[C]}(u)} > \alpha^{-1}$.

Hence, the probability of selecting, in each step, a set of nodes leading to an optimal solution is at least

$$\prod_{i=1}^{\ell} \alpha^{-OPT_i} = \alpha^{-OPT},$$

where ℓ is the number of iterations of the algorithm to produce the vertex cover. $\square$

4.3.3 Experimental Results

Including the 4 different variants of the vertex-based branching approach, we test the 7 different initialization heuristics and compare their performance on different test cases.

There are some well-known MVC benchmarks which have been used to evaluate the performance of different MVC solvers. One of the benchmarks is the DIMACS benchmark set. The instances in the benchmark set are designed to be hard MVC problems.

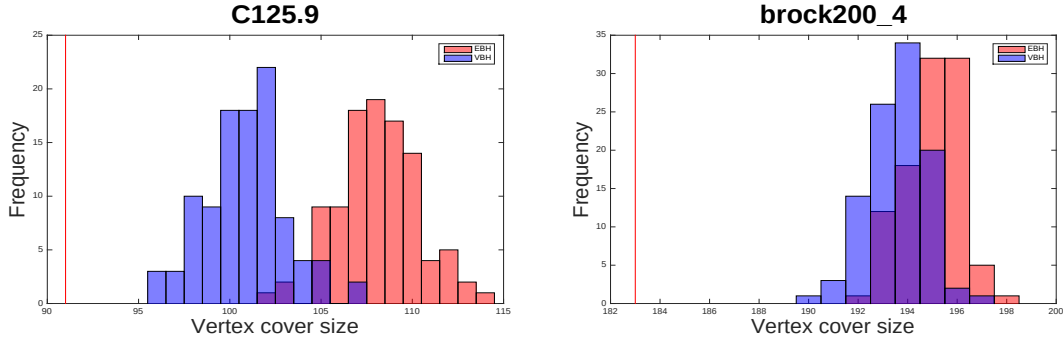


FIGURE 4.1: The histograms show the frequency that each algorithm gets the initial vertex cover of certain size for two sample instances from *DIMACS* benchmark set. The optimal vertex cover size of each instance is indicated with red vertical line in each figure.

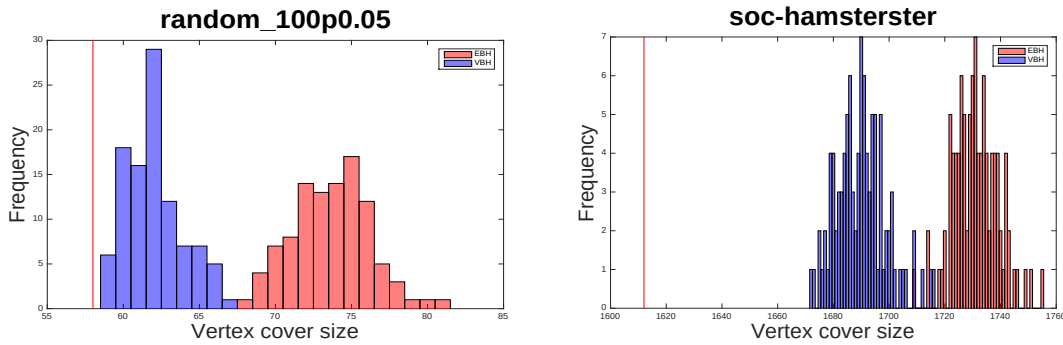


FIGURE 4.2: The histograms show the frequency that each algorithm gets the initial vertex cover of certain size for sample instances that randomly generated or from real world graph. The optimal vertex cover size of each instance is indicated as a red vertical line in each figure.

The *DIMACS* benchmark is a set of challenge problems which comes from the Second *DIMACS* Implementation Challenge for Maximum Clique, Graph Coloring and Satisfiability [83]. The original Max Clique problems are converted to complement graphs to serve as MVC problems.

Besides the benchmark problems, we also test these algorithms on some undirected random graphs and real world graphs. The undirected random graphs are generated with a pre-defined instance size and selection rate of edges. An edge between any two nodes is added to the graph with a certain pre-defined probability. In [136], there are a number of real world graphs with various numbers of vertices and edges. The sample graphs are selected from the undirected unweighted graphs.

All of the algorithms are implemented in JAVA and each of the programs is executed for 101 independent runs on each instance to obtain the statistics.

We first conduct a comparison between the results from the two branching heuristics. In Algorithm 4.2, the *uniform* variant is chosen for the first experiment.

The histograms in Figure 4.2 are achieved by comparing the vertex cover sizes that the two algorithms get from running on four instances in different categories. The distribution of

Instance				EBH					VBH				
Name	V	E	OPT	min	Q1	median	Q3	max	min	Q1	median	Q3	max
random_50p0.1	50	117	28	31	35	36	37	40	28	29	30	31	33
random_50p0.1-2	50	139	31	34	37	38	39	43	31	32	33	34	36
random_100p0.05	100	288	58	68	72	74	75	81	59	61	62	63	67
random_100p0.05-2	100	261	58	67	71	73	75	79	58	60	61	62	66
random_500p0.01	500	1206	284	344	353	357	362	371	292	296	298	301	308
random_500p0.01-2	500	1282	284	344	358	362	365	372	290	298	300	302	308
soc-hamsterster	2426	16630	1612	1709	1726	1731	1737	1755	1672	1684	1690	1695	1716
soc-wiki-Vote	889	2914	406	486	501	508	513	532	406	406	407	409	412
web-edu	3031	6474	1451	1742	1765	1771	1780	1793	1451	1452	1453	1454	1457
web-google	1299	2773	498	582	596	604	611	632	501	506	508	509	517
bio-celegans	453	2025	249	286	293	298	300	306	254	260	263	266	277
bio-yeast	1458	1948	456	583	608	618	626	656	456	459	460	462	468
brock200_4	200	6811	183	192	194	195	196	198	190	193	194	194	197
brock400_4	400	20035	367	390	392	393	394	396	387	390	391	392	395
brock800_4	800	111957	774	792	794	795	796	798	792	793	794	794	797
C125.9	125	787	91	102	107	108	110	114	96	100	101	102	107
C250.9	250	3141	206	227	231	232	234	238	222	225	226	228	232
C500.9	500	12418	443	474	479	481	483	487	467	474	476	477	480

TABLE 4.1: Experimental results on instances comparing the statistics between Algorithm 4.1 and 4.2.

the solutions obtained in 101 independent runs is visualized with the histograms. In the first histogram and those lying in the second row, it is clear that the vertex-based initialization generates smaller solutions for these three instances. For the instance brock200_4 from DIMACS benchmark set, the vertex-based approach has higher probability to generate better initial solutions than its edge-based counterpart.

Table 4.1 shows the five-number summary for each ranked set of 101 results obtained for the two branching approaches testing on specific instance. The quartiles present the quality of initial solutions produced by Algorithm 4.1 and 4.2. From Table 4.1, the initial solutions of real world graphs generated by Algorithm 4.2 are all smaller than those from Algorithm 4.1. For the graphs from random and DIMACS benchmark set, the vertex-based approach can give better initial solutions for most times. Moreover, the vertex-based approach is able to generate solutions which are already globally optimal for some of the instances in random and real world categories.

Then a comparison between results from all seven different approaches is conducted. Some statistics are included as box plots in Figure 4.3. The behaviours of different algorithms have some relationship with the structure of the problem instance. The average degrees of nodes for the instances in DIMACS are higher than those of the randomly generated instances and real world graphs, which implies the DIMACS benchmarks are much denser graphs.

The performance of the edge-based initialization process is always not good comparing to the other approaches. In most cases, it ends up with solution sets much larger than those from other approaches. The initial solution sets found by the vertex-based branching with *mindeg* variant have strong relationship with the average degree of nodes. For Figure 4.3, it is clear that the algorithm terminates with the worst initial solutions for sample instances in DIMACS set among the seven initial processes. The resulting initial sets from Algorithm 4.4 are moderate in size for most cases. The vertex-based approach starting from the whole set performs better for the instances with higher average degree than in loosely connected graphs.

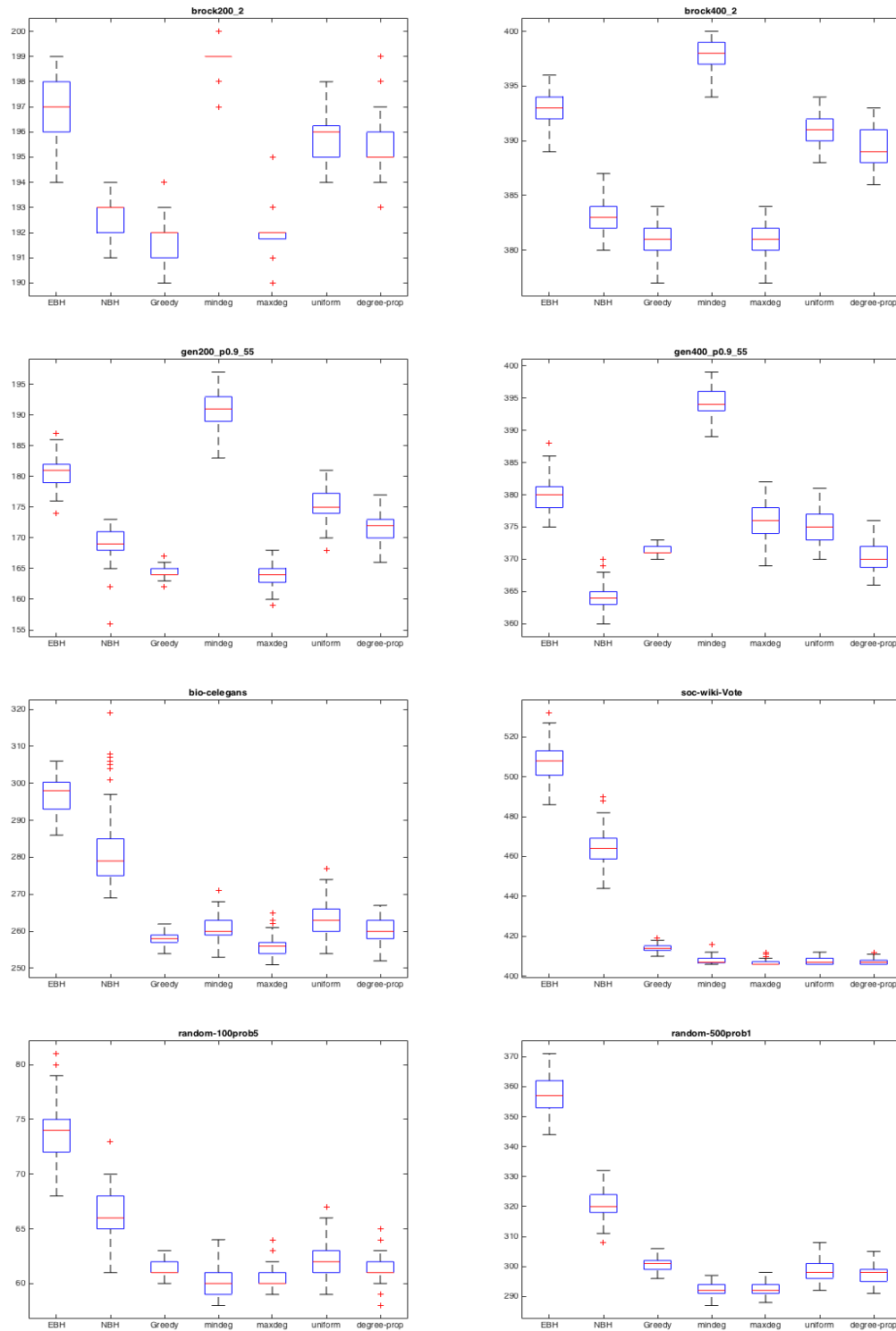


FIGURE 4.3: The box plots show the distribution of the results from 101 independent runs of seven different initialization approach on different instances. EBH, NBH and Greedy represent the results from Algorithm 4.1, 4.4 and 4.3 respectively. mindeg, maxdeg, uniform and degree-prop refer to the four variants of Algorithm 4.2. The y-axis represents the size of initial solution set.

Algorithm 4.5: Edge-based Local Search

```

1 Let  $C$  be an initial vertex cover represented as a list;
2 repeat
3   Choose a node  $v \in C$  uniformly at random and set  $C := C \setminus v$ ;
4   while (( $C$  is not a vertex cover of  $G$ ) and (not termination condition)) do
5     Choose the first node  $v$  of  $C$  and set  $C := C \setminus v$ ;
6     Let  $e = \{u, v\}$  be a random uncovered edge, i.e.,  $e \in G[C]$ ;
7     with probability  $1/2$  do
8        $C := C \cup \{u\}$ 
9     else
10       $C := C \cup \{v\}$ 
11 until termination condition;
12 Return  $C$ ;

```

Among the four different variants of the vertex-based branching approach, the behaviours of *uniform* and *degree-prop* are similar. The *mindeg* variant works the worst among these four, especially for the hard instances in DIMACS benchmark set.

The greedy approach is the common initialization approach for many MVC solvers. From the experimental results, the performance of the vertex-based branching with *maxdeg* variant is similar to that of the greedy algorithm and in some cases better than that. Both of these two approaches provide good initial solutions comparing to other approaches in most cases.

4.4 Two Local Search Algorithms for MVC

We now introduce local search algorithms that make use of the aforementioned branching ideas. Both local search algorithms work with an ordered list C representing a set of nodes and adding a node to C in both algorithms always means appending it to the end of the list. The two algorithms are then compared in solving sample MVC problems from benchmark sets and real world graphs mentioned in the previous section.

4.4.1 Edge-based Local Search Algorithm

The edge-based local search algorithm (see Algorithm 4.5) can be seen as a simplified version of one of the most successful approaches for solving the MVC problem, namely NuMVC [16]. It starts with a vertex cover of size $(k + 1)$ and tries to achieve a smaller vertex cover of size k by removing one node. If this step violates the property of a vertex cover, it removes an additional node, picks an uncovered edge and adds one of its two nodes uniformly at random. After a vertex cover of size k is obtained, it continues the process to search for a vertex cover of size $(k - 1)$ until the termination criterion is reached.

In the following, an upper bound is proved based on the number of steps of edge-based local search to find a vertex cover of size k . For our analysis, we partition the run of edge-based local search into distinct phases of length k which consist of k iterations of the while-loop.

Theorem 4.3. *For all r with $0 \leq r \leq OPT$, the edge-based local search finds a vertex cover of size $k := 2 OPT - r$ after (expected) at most 2^{r+1} phases of length k .*

Proof. We investigate the probability that during k steps of the while-loop a vertex cover has been found at least once. We call this a *success* during a phase of k steps. Let C^* be a vertex cover of size OPT . As C^* is a vertex cover, it contains for each edge $e \in E$ at least one vertex. Consider an edge $e = \{u, v\}$. At each iteration, a vertex $z \in C^*$ is picked with probability at least $1/2$ and each node of C^* is picked at most once as only uncovered edges are chosen. The expected number of distinct vertices contained in C^* during a phase of k steps is therefore at least $k/2 = (2 OPT - r)/2$. The probability that during the first r steps only nodes of C^* are picked is at least 2^{-r} . The expected number of nodes of C^* picked in the remaining $(2 OPT - 2r)$ steps (before a vertex cover is reached) is at least $(OPT - r)$. Furthermore, it is at least $(OPT - r)$ with probability $1/2$. Hence, the algorithm picks all OPT nodes during a phase of $k = 2 OPT - r$ steps with probability at least $2^{-(r+1)}$. The expected number of phases of length k needed to find a vertex cover is therefore at most 2^{r+1} . $\square$

4.4.2 Vertex-based Local Search Algorithm

The vertex-based branching approach is used to design a vertex-based local search algorithm (see Algorithm 4.6). This approach searches for a vertex cover after removing a node together with all its neighbours. Afterwards, it tries to obtain a new vertex cover by picking a random node of minimum degree in the graph consisting of currently all uncovered edges at that time. Based on the degree of this node, different degree rules are applied with the already introduced biased probabilities. The last step is iterated until a vertex cover is found again.

4.4.3 Experimental Analysis

We test Algorithm 4.5 and 4.6 on some sample instances to evaluate their performance. Both algorithms are given an initial vertex cover produced by Algorithm 4.1 and the cut off generation is set to 100 000. Both algorithms are implemented in JAVA and the performance is measured by the number of iterations it takes for the algorithm to make improvements.

Figure 4.4 shows the improvement of the two algorithms on example instances over iterations. $|C| - OPT$ denotes the size difference between the best solution so far and the globally optimal solution. The stairstep lines are drawn for three independent runs for each instance and algorithm. The vertex-based heuristic makes significant improvements before 2 000 generations for these three instances from the observation of the solid lines while the solution of

Algorithm 4.6: Vertex-based Local Search

```

1 Set  $\alpha := 1.4656$ ;
2 Let  $C$  be an initial vertex cover represented as a list;
3 repeat
4   Choose the first node  $v$  of  $C$  and set  $C := C \setminus N_G^2[v]$ ;
5   repeat
6     Let  $u$  be a random node with  $\deg_{G[C]}(u) = \text{mindeg}(G[C])$ ;
7     if  $\deg_{G[C]}(u) = 1$  then
8        $C := C \cup N_{G[C]}[u]$ ; /* degree 1 rule */
9     else if  $\deg_{G[C]}(u) = 2$  then
10      Let  $v, w \in V$  such that  $N_{G[C]}[u] = \{v, w\}$ ;
11      with probability  $\alpha^{-|N_{G[C]}[v] \cup N_{G[C]}[w]|}$  do
12         $C := C \cup N_{G[C]}[v] \cup N_{G[C]}[w]$ 
13      else
14         $C := C \cup N_{G[C]}[u]$ ; /* degree 2 rule */
15      else
16        with probability  $\alpha^{-\deg_{G[C]}(u)}$  do
17           $C := C \cup N_{G[C]}[u]$ 
18        else
19           $C := C \cup \{u\}$ ; /* degree  $\geq 3$  rule */
20    until  $C$  is a vertex cover of  $G$  (or termination condition);
21 until termination condition;
22 Return  $C$ ;

```

edge-based heuristic does not improve much until 100 000 which is the cutoff bound. For the random graphs, the vertex-based approach is able to find a global optimum before 10 000 iterations whereas the edge-based heuristic does not reach the optimal solution before 100 000 iterations.

More results are shown in Table 4.2. The average best vertex cover sizes at certain number of iterations from 10 independent runs of these two algorithms on a certain MVC problem are listed in the table. From the statistics in Table 4.2, vertex-based approach produces better results for 15, 15, 16 and 16 out of the 17 instances after 10 000, 50 000, 100 000 and 200 000 iterations, respectively. Moreover, Algorithm 4.6 has a success rate of 100% in solving the 8 instances from different categories.

4.5 Conclusion

This chapter focuses on the discussion of heuristic algorithms for the MVC problem which is an NP-hard combinatorial optimization problem. There have been many studies into the MVC problem from both theoretical and practical perspectives. Many exact and heuristic algorithms have been designed for solving MVC. The local search algorithms for MVC problem attract more and more attention due to its success in solving many combinatorial

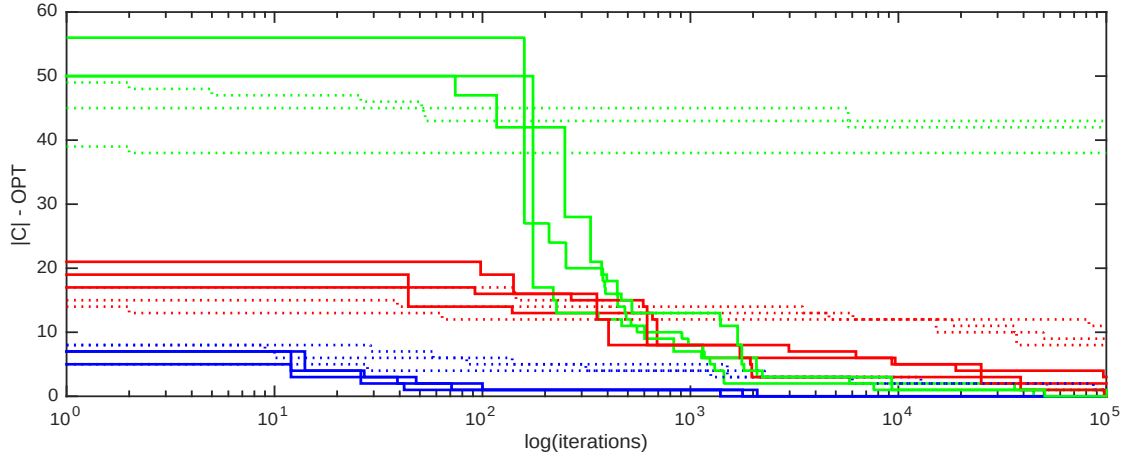


FIGURE 4.4: The improvement of both algorithms in three example instances over iterations. The lines in blue, red and green color represent an independent run on the instance random-50prob10, C125.9 and bio-celegans, respectively. The dotted lines and solid lines denote the results from Algorithm 4.5 and 4.6.

Instance		EBH				VBH			
Name	OPT	10 000	50 000	100 000	200 000	10 000	50 000	100 000	200 000
random_50p0.1	28	29.8±0.632	29.4±0.516	28.9±0.316	28.8±0.422	28.0±0	28.0±0	28.0±0	28.0±0
random_50p0.1-2	31	33.0±0.471	32.6±0.516	32.1±0.316	32.0±0	31.0±0	31.0±0	31.0±0	31.0±0
random_100p0.05	58	66.7±1.160	65.4±0.843	65.1±0.568	64.8±0.422	58.0±0	58.0±0	58.0±0	58.0±0
random_100p0.05-2	58	66.4±1.430	64.8±0.632	64.3±1.160	64.0±0.943	58.0±0	58.0±0	58.0±0	58.0±0
random_500p0.01	284	351.3±3.713	348.8±2.251	348.2±1.932	346.6±1.506	286.4±1.350	284.9±0.568	284.4±0.516	284.4±0.516
random_500p0.01-2	284	357.0±2.828	354.9±1.729	353.1±1.729	352.3±1.703	286.3±1.703	284.2±0.422	284.2±0.422	284.0±0
bio-celegans	249	291.4±2.503	290.7±2.214	290.0±1.764	289.8±1.764	250.7±0.675	249.7±0.483	249.3±0.483	249.3±0.483
bio-diseasome	285	316.2±4.022	314.5±3.100	314.5±3.100	313.3±2.710	288.9±0.675	287.3±0.483	287.0±0.738	286.6±0.483
soc-dolphins	34	36.3±0.483	35.7±0.483	35.4±0.516	34.9±0.316	34.0±0	34.0±0	34.0±0	34.0±0
soc-wiki-Vote	406	502.2±6.647	502.2±6.647	502.2±6.647	502.2±6.647	406.2±0.422	406.0±0	406.0±0	406.0±0
ca-netscience	214	243.9±2.079	241.1±1.370	240.3±1.418	238.7±0.949	216.7±0.675	215.7±0.483	215.1±0.738	214.7±0.483
ca-Erdos992	461	819.1±19.762	808.3±11.186	801.2±8.509	794.9±5.195	461.0±0	461.0±0	461.0±0	461.0±0
C125.9	91	102.7±0.675	101.1±1.197	100.5±0.850	100.4±0.699	95.3±0.823	93.3±1.059	92.8±0.789	92.8±0.789
C250.9	206	228.2±0.919	226.8±0.632	226.3±0.675	225.6±0.843	232.5±2.224	231.5±1.841	231.2±1.476	230.6±1.647
MANN_a27	252	261.0±0	260.8±0.422	260.4±0.516	260.1±0.316	252.9±0.316	252.6±0.516	252.3±0.483	252.1±0.316
MANN_a45	690	705.0±0	705.0±0	705.0±0	705.0±0	701.6±1.897	694.2±0.632	693.3±0.823	692.7±0.675
MANN_a81	2 221	2 241.0±0	2 241.0±0	2 241.0±0	2 241.0±0	2 241.4±0.699	2 241.1±0.568	2 239.0±2.000	2 235.1±1.524

TABLE 4.2: Performance comparison between Algorithm 4.5 and 4.6 on some sample instances. The average vertex cover size is listed after running each algorithm for certain number of iterations.

optimization problems. In this chapter, we first conduct a comparison between different initialization processes for MVC problem. Then we investigate the two fixed-parameter branching algorithms with experimental results.

For many local search algorithms for MVC, the greedy initialization process is their first choice. We investigate the performance of different initialization processes on various problem instances. The randomized initialization strategy coming from a well-known fixed parameter branching approach shows good performance with certain variant. This research provides an insight that the different behaviours of initialization processes may lead to different starting points for MVC algorithms.

The two fixed-parameter local search algorithms for the MVC problem with different branching rules show different behaviours in solving MVC instances. We demonstrate how the vertex-based branching rules can be incorporated into a vertex-based local search algorithm

and show that this approach leads to better results on random generated graphs and social networks than the edge-based local search which is equivalent to the core component of the state-of-the-art local search algorithm NuMVC.

The music is not in the notes, but in the silence between.

Wolfgang Amadeus Mozart

CHAPTER 5

SCALING UP LOCAL SEARCH FOR MVC IN MASSIVE GRAPHS

5.1 Introduction

Although local search algorithms have shown to be one of the successful approaches for many combinatorial optimization problems including MVC [1, 78], they often suffer from the problem of getting trapped in local optimal solutions. Since often these approaches include a random initialization or other random components, running an algorithm several times on a given instance might help with finding a global optimum. However, if the probability of getting stuck in a local optimum is high, then even repeated runs might not help to evade the local optima.

In this chapter, we present a new approach for scaling up existing high-performance local search solvers in order to perform well on massive graphs. Our approach builds on the assumption that massive graphs are composed of different (hidden) substructures. Substructures often occur in large social network graphs as social networks usually consist of (loosely connected) sub-communities. In massive graphs the issue of local optima might occur in the different substructures of the given problem instance and having a large number of these substructures where an algorithm even just fails with a small probability might make it very hard for local search approaches to obtain the optimal solution. In this chapter, we propose a simple parallel kernelization approach that builds on theoretical investigations regarding substructures on massive graphs.

Kernelization approaches have been shown to be very effective in algorithms which have a good performance guarantee [41, 31]. The key idea is to pre-process a given problem instance by making optimal decisions on easy parts of the given input such that the overall problem instance is reduced. Afterwards, the main effort is spent on the reduced instance which is called the kernel. There are several kernelization techniques available for the MVC problem which perform well if the number of vertices in an optimal solution is small. However, the applicability to difficult instances which are usually large dense graphs is limited as the pre-processing does not significantly reduce the problem instance size.

In this chapter, we present a new way of reducing the problem instance size by parallel kernelization (note that this is not the kernelization in the theoretical sense). This approach uses existing local search solvers to deal with massive graphs. The key idea is to do μ parallel runs of such a solver and reduce the given instance by fixing components that have been selected in all μ runs and reducing the instance afterwards. The resulting *reduced* instance is then solved by an additional run of the local search solver and the combined result is returned as the final solution.

The approach can be applied in solving many combinatorial optimization problems. We consider the MVC problem as an example to illustrate the effectiveness of our approach. Popular local search approaches for tackling MVC include PLS [128], NuMVC [16], TwMVC [15], COVER [135]. These approaches are usually evaluated on standard benchmarks and (in more recent years on) massive real world graphs. We take NuMVC as the baseline local search solver for our new kernelization approach. This algorithm belongs to the best-performing approaches for MVC and has the advantage that it does not require much effort in parameter tuning for different types of benchmark instances. Our experimental results show that our new kernelization technique does not do any harm on instances where NuMVC is already performing well, moreover it improves the results on graphs combined of different copies of the benchmark problems. Furthermore, on social network graphs the kernelization reduces the massive graphs significantly such that only 10% – 20% of the vertices remain in the kernelized instance; hence, the kernelizing algorithm significantly outperforms the plain version of NuMVC on most massive real world network graphs considered in our experimental investigations.

The outline of this chapter is as follows. In Section 5.2, the theoretical motivation of the parallel kernelization technique is discussed in details. The resulting local search approach from parallel kernelization of MVC problem is included as Section 5.3. After that we present some experimental results to evaluate the performance of our new approach in Section 5.4 and 5.5 on classical benchmark problems and massive social network graphs, respectively. The chapter is finished with some concluding remarks in 5.6.

5.2 Substructures in Massive Graphs

Massive graphs originating for example from social networks consist of a large number of vertices and edges. Our approach builds on the assumption that these graphs are composed of different substructures which on their own and at a small scale would not be hard to handle by current local search approaches. This is for example the case for social networks which are composed of different communities. The difficulty arises through the composition of substructures that are not known to the algorithm and are hard to extract from the given instances.

Assume that a randomly initialized local search algorithm executes on an instance that consists of different subparts s_i , $1 \leq i \leq k$, where each part s_i has a probability p_i of failing to obtain the optimal sub-solution independently of the other components. Then the probability of obtaining the optimal solution is

$$\prod_{i=1}^k (1 - p_i).$$

Even if there is only a constant probability $p' = \min_{i=1}^k p_i$, $0 < p' < 1$ of failing in each of the k components, the probability that the local search algorithm solves the overall instance would be exponentially small in k , i.e. the algorithm only succeeds with probability

$$\prod_{i=1}^k (1 - p_i) \leq \prod_{i=1}^k (1 - p') = (1 - p')^k \approx e^{-p' \cdot k}. \quad (5.1)$$

In the kernelization, the local search algorithm is executed μ times independently with random initialization process. After some time t_1 for each of these runs, we stop the algorithm. After all μ solutions are computed, we *freeze* the setting for all those components that are set the same way in all μ runs. As the last step, the local search algorithm executes on the *reduced instances* with the frozen components removed.

Consider a component s_i again where the probability of failing is p_i . The probability that a single run of the algorithm obtains the optimal solution for this component is $(1 - p_i)$ and the probability that μ random runs identify an optimal solution is $(1 - p_i)^\mu$. As long as the failure probability p_i is only a small constant and μ is not large, this term is still a constant that is sufficiently large, which shows that the kernelization will likely be successful as well. Let $|s_i|$ be the size of component s_i . Furthermore, we assume that the whole instance s is composed of the k subcomponents and we have $|s| = \sum_{i=1}^k |s_i|$.

The expected decrease in size of the original problem consisting of the components s_i is given by

$$\sum_{i=1}^k (1 - p_i)^\mu |s_i|.$$

Assuming $\hat{p} = \max_{i=1}^k p_i$, then we get

$$\sum_{i=1}^k (1 - p_i)^\mu |s_i| \geq (1 - \hat{p})^\mu \sum_{i=1}^k |s_i| = (1 - \hat{p})^\mu \cdot |s|. \quad (5.2)$$

We now consider the probability that one of the different components has not achieved an optimal sub-solution in at least one of the μ runs. In such a case our algorithm could potentially reduce the instance and fix vertices of that component which do not belong to an optimal solution. In this case, the kernelization step would fail and prevent us from obtaining the overall optimal solution.

Consider component s_i . The probability that all μ runs of the solver do not obtain the optimal sub-solution for this component is p_i^μ . The probability that at least one of them obtains the optimal sub-solution is therefore at least

$$1 - p_i^\mu$$

and the probability that for each component there is at least one run where the optimal sub-solution is obtained is therefore at least

$$\prod_{i=1}^k (1 - p_i^\mu) \geq (1 - \hat{p}^\mu)^k \approx e^{-\hat{p}^\mu \cdot k}. \quad (5.3)$$

As an example, assume that the probability of the original approach failing on each subcomponent is 10%, $\mu = 3$, and $k = 50$. Then the expected reduction according to Equation 5.2 is $(1 - 0.1)^3 \cdot |s| = 0.729 \cdot |s|$, i.e. the resulting instance has only 27.1% of the original number of vertices. The probability of not failing in the reduction step according to Equation 5.3 is $(1 - 0.1^3)^k = 0.999^k$, whereas the probability of a single run of the original approach not failing in at least one component according to Equation 5.1 is $(1 - 0.1)^k = 0.9^k$. For $k = 50$ we get a probability of not failing in the kernelization step of $0.999^{50} \approx 0.95$ and a probability of not failing in the original algorithm of $0.9^{50} \approx 0.005$.

The user can control μ and from our calculations it can be observed that there is a trade-off between reducing the number of vertices and the probability of fixing the wrong vertices in at least one of these components in dependence of μ .

5.3 Parallel Kernelization for MVC

We now show how to use the ideas discussed in the previous section in an algorithmic sense. As mentioned previously, our approach assumes that there is already a good local search solver for the given problem P for small to medium size instances. Our goal is to use parallel kernelization to make it work for massive instances. We take the well-known

Algorithm 5.1: Local Search with Parallel Kernelization

-
- 1 Initialize P with μ solutions after μ different independent runs of MVC solver with cutoff time t_1 .
 - 2 Let set V_a be the set of vertices which are selected by all solutions in P .
 - 3 Construct an instance I with vertices $v \notin V_a$ and edges which are not adjacent to any vertex in V_a .
 - 4 Run MVC solver on instance I with cutoff time t_2 to get a minimum vertex cover V_s .
 - 5 Construct the final solution $V_c = V_a \cup V_s$.
-

NP-hard MVC problem as an example problem, but expect that our approach is applicable to a wide range of other problems as well.

The main idea is to kernelize the vertex set and form a smaller instance for the MVC solver to solve. Firstly the MVC solver is run μ times on the given graph $G = (V, E)$ with a cutoff time t_1 for each run to achieve a set of μ solutions. The vertices which are selected in all μ solutions are added to a separate set V_a and the edges that are covered by the vertices of V_a are removed from the edge set. The new instance $G' = (V', E')$ is formed by the vertices that are not selected in all μ solutions and the edge set after deletion, i.e. we have $V' = V \setminus V_a$ and $E' = E \setminus \{e \in E \mid e \cap V_a \neq \emptyset\}$. The MVC solver is run on the new instance G' to obtain a minimum vertex cover V_s . The overall solution for the original graph G is

$$V_c = V_a \cup V_s$$

and consists of the set of vertices which are selected in all μ initial solutions and the minimum vertex cover achieved by the MVC solver running on the new instance G' . It should be noted that it is crucial that the cutoff time t_1 allows the μ runs to obtain at least nearly locally optimal solutions. A detailed description of our approach is given in Algorithm 5.1.

For our experimental investigations we use NuMVC [16] as the MVC solver. This is one of the best performing local search approaches for MVC and has the advantage over TwMVC [15] that it does not require parameter tuning for different types of benchmark instances.

5.4 Experimental Results on DIMACS and BHOSLIB Benchmarks

In this section, we discuss our experiments carried out with an implementation of Algorithm 5.1 compared with single run of NuMVC. The total time budget that both algorithms can use is the same.

The NuMVC is open-source and implemented in C++. We compile the NuMVC source code with g++ with '-O2' option. The parameter setting follows what is reported in [16].

Taking NuMVC as the MVC solver in Algorithm 5.1, we refer to this new approach to solve MVC as NuMVC-PK, since it is strongly based on the original NuMVC program. We

Name	OPT	NuMVC-PK			NuMVC		
		$VC_{\min}$	VC_{avg}	sr	$VC_{\min}$	VC_{avg}	sr
frb40-19-1	720	720	720.0	1.0	720	720.0	1.0
frb40-19-2	720	720	720.0	1.0	720	720.0	1.0
frb40-19-3	720	720	720.0	1.0	720	720.0	1.0
frb40-19-4	720	720	720.0	1.0	720	720.0	1.0
frb40-19-5	720	720	720.0	1.0	720	720.0	1.0
frb45-21-1	900	900	900.0	1.0	900	900.0	1.0
frb45-21-2	900	900	900.0	1.0	900	900.0	1.0
frb45-21-3	900	900	900.0	1.0	900	900.0	1.0
frb45-21-4	900	900	900.0	1.0	900	900.0	1.0
frb45-21-5	900	900	900.0	1.0	900	900.0	1.0
brock400_2	371	371	372.2	0.7	371	372.2	0.7
brock400_4	367	367	367.0	1.0	367	367.0	1.0
brock800_2	776	779	779.0	0.0	779	779.0	0.0
brock800_4	774	779	779.0	0.0	779	779.0	0.0
C2000.9	1,920	1,921	1,921.8	0.2	1,921	1,921.6	0.4
C4000.5	3,982	3,982	3,982.0	1.0	3,982	3,982.0	1.0
MANN_a45	690	690	690.0	1.0	690	690.0	1.0
MANN_a81	2,221	2,222	2,222.8	0.0	2,221	2,222.6	0.2

TABLE 5.1: This table contains the results from NuMVC-PK and NuMVC on the BHOSLIB and DIMACS benchmark. Column sr refers to the success rate. The cutoff time of single NuMVC is set to 1,000 seconds. The parameters for NuMVC-PK are set to $\mu = 3$, $t_1 = 200$ and $t_2 = 400$.

then conduct experiments to investigate the behaviour of NuMVC-PK comparing to simple restart of NuMVC.

Each experiment on a certain instance for each algorithm is executed 10 times in order to gather statistics. The cutoff time for the initial runs of NuMVC-PK is set based on initial experimental investigations on the different classes of instances considered. Based on our theoretical investigations carried out in Section 5.2, it is important that each of the μ runs obtains at least nearly locally optimal solutions for the given problem. This implies that a too small cutoff time t_1 might have detrimental effects.

All of the experiments are executed on a machine with two Intel(R) Xeon(R) E5-2650 2.00GHz CPUs and 64GByte RAM; note that the program uses only a single core. The memory consumption depends on the instance size and the MVC solver.

5.4.1 Results for Original DIMACS and BHOSLIB Benchmarks

There are some well-known MVC benchmark sets which have been used to evaluate the performance of different MVC solvers. Two of the benchmark sets are the *DIMACS* and the *BHOSLIB* benchmark sets. The *DIMACS* benchmark is introduced in previous chapter.

The *BHOSLIB* (Benchmarks with Hidden Optimum Solutions) problems are generated from translating the binary Boolean Satisfiability problems randomly generated based on the model RB [164]. These instances have been proved to be hard to solve, both theoretically and practically.

Most *BHOSLIB* and *DIMACS* instances are easily solved with good success rates by NuMVC [16]. Table 5.1 shows the comparison between the results from NuMVC-PK and NuMVC on some

Name	Instance			NuMVC-PK				NuMVC		Comparison		
	OPT	V	E	V'	E'	$VC_{\min}$	VC_{avg}	$VC_{\min}$	VC_{avg}	$\Delta VC_{\min}$	p -value	
frb40-19-1_10	7,200	7,600	413,140	577	1,375	7,200	7,200.0±0	7,200	7,200.0±0	0	n/a	no diff.
frb40-19-2_10	7,200	7,600	412,630	1,446	11,543	7,200	7,202.5±1.581	7,205	7,206.0±0.667	5	0.0002***	better
frb40-19-3_10	7,200	7,600	410,950	1,077	6,257	7,200	7,200.0±0	7,201	7,201.9±0.738	1	0.0001***	better
frb40-19-4_10	7,200	7,600	416,050	1,317	9,045	7,200	7,201.1±0.876	7,202	7,202.9±0.876	2	0.0011**	better
frb40-19-5_10	7,200	7,600	416,190	1,524	12,359	7,202	7,204.0±1.054	7,205	7,206.2±0.789	3	0.0004***	better
frb45-21-1_10	9,000	9,450	591,860	1,431	9,862	9,000	9,000.6±0.843	9,001	9,002.0±0.943	1	0.0051**	better
frb45-21-2_10	9,000	9,450	586,240	1,815	16,279	9,000	9,001.4±1.350	9,001	9,002.8±1.229	1	0.0305*	better
frb45-21-3_10	9,000	9,450	582,450	1,697	14,237	9,002	9,004.3±1.703	9,004	9,006.0±1.414	2	0.0335*	better
frb45-21-4_10	9,000	9,450	585,490	1,557	11,149	9,000	9,001.3±1.160	9,001	9,003.4±1.265	1	0.0029**	better
frb45-21-5_10	9,000	9,450	585,790	1,589	12,449	9,003	9,003.2±0.632	9,005	9,005.4±0.699	3	0.0001***	better

TABLE 5.2: This table contains instances that have been tested on, which are generated by duplicating one existing hard instance in BHOSLIB benchmark. The instance name contains the name of original instance and the number of copies. The cutoff time of single NuMVC is set to 3,000 seconds. The parameters for NuMVC-PK are set to $\mu = 5$, $t_1 = 500$ and $t_2 = 500$.

* significant at $p < 0.05$, ** significant at $p < 0.01$, *** significant at $p < 0.001$.

BHOSLIB and DIMACS benchmarks. For most of the instances, both algorithms have good success rate, and NuMVC-PK does not affect the performance of NuMVC.

5.4.2 Results for Combined DIMACS and BHOSLIB Benchmarks

Since the well-known benchmark sets BHOSLIB and DIMACS are designed to be hard problems but can be solved in short time by single run of NuMVC [16], we propose some simple combinations of these existing benchmarks as new test cases. These instances serve as very simple first test cases for our kernelization method. The new instances are composed by several sub-graphs and large in size of both vertices and edges.

In particular, we construct new instances by considering independent copies of an existing instance. Each single copy is easy to be solved by the MVC solver, while the combined instance is much harder to be solved.

Some examples of this kind of instances are given in Table 5.2. The original instances are selected from the BHOSLIB benchmark set; the last number in the instance name after the underscore denotes the number of copies of the given instance indicated by the first part of the instance name. Although the original instances can be solved by NuMVC in reasonable time, it takes much longer for NuMVC to solve the multiplied new instances. NuMVC may get trapped in local optima which are far away from the global optima in search space.

Table 5.2 shows the comparison between results from NuMVC-PK and single run of NuMVC. Each instance has been tested 10 times to get the minimum vertex cover found, the average vertex cover size and some statistics for analysis. The basic information about the instances is included in Table 5.2 as column 'Instance'. The 'OPT' column stores the optimal (or minimum known) vertex cover size. The numbers in the |V| and |E| columns are the numbers of vertices and edges in the corresponding instances. NuMVC-PK is executed with parameters $\mu = 5$, $t_1 = 500$ and $t_2 = 500$, which means 5 independent runs of NuMVC to get initial solution sets after 500 seconds and the NuMVC is run for another 500 seconds on the newly

Name	Instance			NuMVC-PK					NuMVC		
	OPT	$ V' $	$ E' $	μ	t_1	t_2	$VC_{\min}$	VC_{avg}	t	$VC_{\min}$	VC_{avg}
frb40-19-123412	4,320	4,560	247,854	5	200	100	4,320	4,320.9	1,100	4,320	4,320.9
frb40-19-422431	4,320	4,560	248,145	31,000	500		4,320	4,320.5	5,000	4,321	4,321.2
frb40-12345-frb45-123	6,300	6,635	382,951	51,000	100		6,300	6,300.4	5,000	6,300	6,302.4
frb45-21-312444	5,400	5,670	351,702	31,000	200		5,400	5,401	3,500	5,401	5401.6

TABLE 5.3: This table contains instances that have been tested on, which are generated by combining different existing hard instance in BHOSLIB benchmark. The last sequence of numbers in the instance name represents the specific instances chosen from the *BHOSLIB* set.

generated instance to achieve the final solution. The information of the generated reduced instance is listed in columns $|V'|$ and $|E'|$, where the number of non-isolated vertices and edges of the new instance are listed. NuMVC is executed for 3 000 seconds to compare with NuMVC-PK, which has the same time budget.

We used the Wilcoxon unpaired signed-rank test on the solutions from different runs of the algorithms on a given instance and the p -value is listed in Table 5.2. The significance of difference between two sets of results is indicated in the table. The difference between the minimum vertex cover found by two approaches is reported in the column of $\Delta VC_{\min}$.

Since *BHOSLIB* benchmark set consists of hard MVC problems, making sure all sub-graphs to be solved to optimality is hard for a single run of NuMVC, which easily gets trapped in some local optimum. On the other hand, NuMVC-PK shrinks the large instance and takes a fresh start on the reduced instance, thereby improving the performance of the local search.

From the results we see that NuMVC-PK is able to reduce the instance size. For the duplicated *BHOSLIB* instances, after 5 runs of NuMVC, the NuMVC-PK generates new instances which keep only 1% to 3% of the edges and 8% to 20% of the vertices. Unlike NuMVC, which usually makes no improvement after 2 000 seconds, NuMVC-PK finds the global optima for 6 out of the 9 instances where NuMVC ends up with local optima after 3 000 seconds in all 10 runs. For these hard instances, an improvement by a couple of vertices can be seen as significant.

Except for independent multiple copies of certain instances, we also try to connect each pair of sub-graphs by a randomly selected edge which make the whole graph a loosely connected graph. We observe similar behaviour as the instances of multiple copies of the same instance.

5.4.3 Results for Combination of Existing Hard Instances

Except for duplicating existing hard instances, we also try combining different instances into one large graph. The single instances are also chosen from the *BHOSLIB* benchmark set. Some of the examples are shown in Table 5.3. The last sequence of numbers in the instance name represents the specific instances chosen from the *BHOSLIB* set, e.g. frb40-12345-frb45-123 refers to a new instance built from different instance in the order of frb40-19-1, frb40-19-2, frb40-19-3, frb40-19-4, frb40-19-5, frb45-21-1, frb45-21-2 and frb45-21-3. The

instances are generated by randomly picking certain number of instances from the *BHOSLIB* instances set. The information about the new instance is shown in Table 5.3 as column OPT, vertices and edges.

The running time for NuMVC is t_3 and the minimum vertex cover size found is listed in column ' $VC_{\min}$ '. Each experiment is run for 10 times to get an average vertex cover size. The parameter setting for NuMVC-PK is listed in Table 5.3 under the column ps , t_1 and t_2 of NuMVC-PK.

Algorithm 5.1 can solve this type of problems with proper parameter settings. The best parameter combination found so far is listed in Table 5.3 in column ps , t_1 and t_2 . The different parameters tried are 3 and 5 for ps ; 200, 500 and 1000 for t_1 . t_2 is set to 2000 for all tests.

Not all combination of instances are listed in the table. Since this type of instances are different in hardness, the parameter setting is decided based on the results of single NuMVC runs. From the results shown in Table 5.3, the NuMVC-PK provides stable results in shorter runtime.

5.5 Experimental Results on Real World Graphs

Now we turn our attention to comparing NuMVC-PK with NuMVC on massive real world graphs as given by [136]. All of these selected graphs are undirected and with a large number of vertices and edges. In contrast to the benchmark sets considered in Section 5.4.2, the global optima of these instances are unknown. The graphs examined are taken from the social networks, collaboration networks and web link (Miscellaneous) networks packages. Some samples are also selected from the *dImacs10* data sets which come from the 10th DIM-CAS implementation challenge [9]. The graphs have numbers of vertices in the range of 15 000 to 2 600 000 and number of edges in the range of 40 000 to 16 000 000.

The experimental results are summarized in Table 5.4 and 5.5. Same as those in Table 5.2, the columns of $|V|$ and $|E|$ provide the brief information of the graphs (number of vertices and edges, respectively). The categories NuMVC-PK and NuMVC present the comparison between results from NuMVC-PK and single run of NuMVC. Since the huge real world graphs are not as complex as the combined *BHOSLIB* instances, we use $\mu = 3$ to get the initial solution set. The minimum vertex cover found in the 10 runs and the average size of the solutions is reported in the table. The standard deviation for each instance is also included to show the stability of the algorithms. NuMVC is run for 1 000 seconds, corresponding to the total budget of NuMVC-PK. Some easy instances which can be easily solved by single run of NuMVC in short running time are omitted from the table since both algorithms have 100% success rate.

For the real world graphs in social networks, collaboration networks and web link networks packages, NuMVC-PK reduces the instance size by more than 90% in number of vertices and 70% in number of edges. The size of the instance is one of the main factors that affect

Name	Instance		NuMVC-PK				NuMVC		Comparison		
	$ V $	$ E $	$ V' $	$ E' $	$VC_{\min}$	VC_{avg}	$VC_{\min}$	VC_{avg}	$\Delta VC_{\min}$	p -value	
soc-BlogCatalog	88,784	2,093,195	5,865	3,687	20,752	20,752.0±0	20,752	20,752.9±0.876	0	0.0057**	better
soc-brightkite	56,739	212,945	9,426	5,858	21,190	21,190.0±0	21,193	21,197.9±2.644	3	0.0001***	better
soc-buzznet	101,163	2,763,066	8,860	6,095	30,614	30,615.4±1.265	30,614	30,615.5±1.269	0	0.9376	no diff.
soc-delicious	536,108	1,365,961	25,728	29,531	85,378	85,397.1±12.315	85,539	85,573.3±29.201	161	0.0002***	better
soc-digg	770,799	5,907,132	16,556	25,111	103,249	103,254.6±3.169	103,314	103,324.0±8.602	65	0.0002***	better
soc-douban	154,908	327,162	288	145	8,685	8,685.0±0	8,685	8,685.0±0	0	n/a	no diff.
soc-epinions	26,588	100,120	4,790	2,917	9,757	9,757.0±0	9,757	9,757.0±0	0	n/a	no diff.
soc-flickr	513,969	3,190,452	65,286	37,019	153,607	153,629.5±14.128	153,338	153,348.8±8.277	-269	0.0018**	worse
soc-flixster	2,523,386	7,918,801	2,229	4,046	96,339	96,342.9±2.846	96,320	96,322.2±1.135	-19	0.0001***	worse
soc-FourSquare	639,014	3,214,986	10,406	7,414	90,110	90,111.6±0.843	90,130	90,135.1±4.012	20	0.0002***	better
soc-gowalla	196,591	950,327	40,406	28,477	84,245	84,251.5±2.759	84,327	84,336.0±5.558	82	0.0002***	better
soc-lastfm	1,191,805	4,519,330	3,830	2,190	78,688	78,688.5±0.527	78,693	78,696.2±1.874	5	0.0001***	better
soc-LiveMocha	104,103	2,193,083	12,603	9,225	43,430	43,433.0±1.333	43,434	43,442.4±4.452	4	0.0003***	better
soc-slashdot	70,068	358,647	10,901	6,668	22,373	22,373.0±0	22,376	22,379.0±2.867	3	0.0001***	better
soc-twitter-follows	404,719	713,319	38	19	2,323	2,323.0±0	2,323	2,323.0±0	0	n/a	no diff.
soc-youtube	495,957	1,936,748	65,904	42,442	146,899	146,907.7±8.042	146,459	146,470.3±5.716	-440	0.0002***	worse
soc-youtube-snap	1,134,890	2,987,624	160,412	87,793	277,511	277,522.3±11.136	278,580	278,613.0±24.449	1,069	0.0002***	better
dImacs10-citationCiteseer	268,495	1,156,647	47,408	35,455	118,172	118,185.9±7.795	118,329	118,344.3±8.512	157	0.0002***	better
dImacs10-coAuthorsCiteseer	227,320	814,134	64,583	38,008	129,193	129,193.0±0	129,193	129,194.5±1.269	0	0.0007***	better
dImacs10-cs4	22,499	43,858	12,860	19,092	13,361	13,367.5±3.274	13,368	13,380.4±6.328	7	0.0007***	better
dImacs10-cti	16,840	48,232	13,124	36,366	8,752	8,778.9±26.793	8,752	8,776.7±38.286	0	0.5012	no diff.
dImacs10-delaunay-n15	32,768	98,274	9,109	10,328	22,444	22,450.0±3.498	22,456	22,460.2±2.348	12	0.0002***	better
dImacs10-delaunay-n16	65,536	196,575	23,830	27,888	44,984	44,992.6±7.245	44,946	44,968.0±12.763	-38	0.0004***	worse
dImacs10-delaunay-n17	131,072	393,176	53,583	66,138	90,294	90,342.7±22.623	90,584	90,616.6±28.320	290	0.0000***	better

TABLE 5.4: Experimental results on instances from some real world graphs about social networks. The cutoff time of the single NuMVC run is set to 1,000 seconds. The parameters for NuMVC-PK are set to $\mu = 3$, $t_1 = 200$ and $t_2 = 400$.

** significant at $p < 0.01$, *** significant at $p < 0.001$.

Name	Instance		NuMVC-PK				NuMVC		Comparison		
	$ V $	$ E $	$ V' $	$ E' $	$VC_{\min}$	VC_{avg}	$VC_{\min}$	VC_{avg}	$\Delta VC_{\min}$	p -value	
ca-citeseer	227,320	814,134	68,049	41,230	129,193	129,193.0±0	129,194	129,194.8±0.919	1	0.0001***	better
ca-coauthors-dblp	540,486	15,245,729	93,319	69,523	472,250	472,257.3±5.832	472,324	472,334.8±7.540	74	0.0002***	better
ca-dblp-2010	226,413	716,460	65,615	39,019	121,969	121,969.5±0.527	121,971	121,974.4±1.955	2	0.0001***	better
ca-dblp-2012	317,080	1,049,866	78,406	43,526	164,951	164,953.2±1.687	164,956	164,958.2±2.486	5	0.0005***	better
ca-MathSciNet	332,689	820,644	78,800	48,359	139,955	139,958.7±2.163	139,981	139,988.1±4.175	26	0.0002***	better
web-arabic-2005	163,598	1,747,269	26,120	19,511	114,444	114,448.1±2.759	114,468	114,475.3±3.529	24	0.0002***	no diff.
web-baidu-baike-related	415,641	3,284,387	67,615	74,066	143,581	143,629.8±26.318	144,155	144,190.2±19.424	574	0.0002***	better
web-google-dir	875,713	5,105,039	117,405	95,026	347,783	347,795.6±12.842	347,771	347,826.4±55.674	-12	0.1403	no diff.
web-it-2004	509,338	7,178,413	71,247	64,798	415,017	415,043.3±21.505	414,861	414,895.6±14.447	-156	0.0002***	worse
web-sk-2005	121,422	334,419	35,409	26,361	58,179	58,181.9±2.424	58,201	58,206.4±3.718	22	0.0002***	better

TABLE 5.5: Experimental results on instances from some real world graphs about collaboration networks. The cutoff time of the single NuMVC run is set to 1,000 seconds. The parameters for NuMVC-PK are set to $\mu = 3$, $t_1 = 300$ and $t_2 = 100$.

** significant at $p < 0.01$, *** significant at $p < 0.001$.

Name	NuMVC-PK				NuMVC	
	t_1	t_2	$VC_{\min}$	VC_{avg}	$VC_{\min}$	VC_{avg}
soc-flickr	200	400	153,607	153,629.5	153,338	153,348.8
	300	100	153,293	153,297.5		
soc-LiveMocha	200	400	43,430	43,433.0	43,434	43,442.4
	300	100	43,429	43,431.6		
soc-youtube	200	400	146,899	146,907.7	146,459	146,470.3
	300	100	146,399	146,404.1		
ca-coauthors-dblp	200	400	472,253	472,261.4	472,324	472,334.8
	300	100	472,250	472,257.3		
ca-dblp-2010	200	400	121,974	121,984.4	121,971	121,974.4
	300	100	121,969	121,969.5		
ca-dblp-2012	200	400	164,996	165,011.2	164,956	164,958.2
	300	100	164,951	164,953.2		
ca-MathSciNet	200	400	139,985	139,989.3	139,981	139,988.1
	300	100	139,955	139,958.7		

TABLE 5.6: Experimental results on instances from some real world graphs with different parameter settings. $\mu = 3$ in all cases.

the performance of the MVC solvers for the real world graphs. The instances after shrinking have less than 200 000 vertices and 100 000 edges. For graphs in *dImacs10* package, the generated instances maintain around 20% vertices and 40% edges in most cases.

Wilcoxon unpaired signed-rank tests are done between each two results sets coming from NuMVC-PK and single run of NuMVC. The p -values of each instances are included in Table 5.4. The tests are done in R environment and based on results from the 10 independent runs in each aspect.

Regarding Tables 5.4 and 5.5, we make the following observations.

- NuMVC-PK finds smaller minimum vertex cover in the ten independent runs than NuMVC in 21 out of the 34 graphs.
- In the 7 graphs where both algorithms find the same minimum vertex cover, there are 2 instances for which NuMVC-PK obtains stable results in 10 runs, which means that NuMVC-PK has a higher success rate in solving these problems.
- There are 6 graphs for which NuMVC-PK is not able to return a better solution than NuMVC. These graphs have the property that they are hard or large instances so local optima are not reached within time t_1 or even 1, 000 seconds.
- There are 8 graphs where NuMVC-PK finds a minimum vertex cover smaller by 50 than NuMVC.

For some large instances, the initialization process of NuMVC is very time consuming. Enough time should be given for NuMVC to get initial solutions at least near the locally optimal solutions. For the same time limit, longer single initial runs are more beneficial than shorter initial runs and longer runs after freezing phase. Therefore, a combination of larger t_1 and smaller t_2 may result in a better solution for these instances. For some instances which does not provide good results with the current parameter setting, different parameters are tried to see whether better results can be achieved.

As an illustration, we show in Table 5.6 that, for some real world massive graphs and a run time limit of 1, 000 seconds, the NuMVC-PK can produce better results for longer initial runs

Name	NuMVC-PK				NuMVC		
	t_1	t_2	$VC_{\min}$	VC_{avg}	t	$VC_{\min}$	VC_{avg}
frb40-19-2_10	500	500	7,200	7,202.5	3,000	7,205	7,206
	1,000	100	7,200	7,201.4	10,000	7,204	7,205.3
frb45-21-3_10	500	500	9,002	9,004.3	3,000	9,004	9,006
	1,000	500	9,002	9,003.9	10,000	9,004	9,006.4
frb45-21-5_10	500	500	9,003	9,003.2	3,000	9,005	9,005.4
	1,000	100	9,000	9,001.5	10,000	9,000	9,003.6

TABLE 5.7: Experimental results on instances from some duplicated benchmark graphs with different parameter settings. $\mu = 3$ in all cases.

(the reduced graph does not require much further work). With parameters $\mu = 3$, $t_1 = 300$ and $t_2 = 100$, NuMVC-PK achieves better solution than NuMVC. Improvement of solving duplicated *BHOSLIB* benchmark instances by different parameter settings can be seen from some examples shown in Table 5.7.

5.6 Remarks and Conclusions

In this section, we have presented a new approach for scaling up local search algorithms for massive graphs. Our approach builds on the theoretical assumption that massive graphs are composed of different substructures which are on their own not hard to be optimized. Our approach is based on parallel kernelization and reduces the given graph by making μ parallel randomized runs of the given local search and fixing components which have been chosen in all different independent runs. The resulting instance is then tackled by an additional run of the local search approach. Considering the MVC problem and the state-of-the-art local search solver NuMVC, we have shown that our parallel kernelization technique is able to reduce standard benchmark graphs and massive real world graphs to about 10 – 20% of their initial sizes. Our approach outperforms the baseline local search algorithm NuMVC in most test cases.

The parallel kernelization approach presented in this paper can be applied to a wide range of combinatorial optimization problems for which well performing local search solvers are available. The whole process can be accelerated by multithread computing. We plan to investigate the application to other problems such as Maximum Clique and Maximum Independent Set in the future.

Nothing in life is to be feared, it is only to be understood.

Marie Curie

CHAPTER 6

DIVERSITY MAXIMIZATION FOR SINGLE-OBJECTIVE PROBLEMS IN DECISION SPACE

6.1 Introduction

Evolutionary Algorithms (EAs) are a class of search algorithms which are widely applied in solving complex problems in various areas such as combinatorial optimization, bioinformatics and engineering [119]. There are a lot of different types of EAs, among which the famous ones are Genetic Algorithms, Genetic Programming, Evolutionary programming and Evolution Strategies as introduced in Chapter 2.

Evolutionary Algorithms usually work with a set of solutions called the population which is evolved during the optimization process. Diversity lies at the heart of population-based EAs and there are many different mechanisms such as crowding and fitness-sharing which have wide applications. A diverse set of individuals is often beneficial for preventing premature convergence to locally optimal solutions from an optimization point of view [99]. On the other hand, from a design point of view, a diverse set of solutions provides different choices to the decision makers.

In this chapter, we present some theoretical analysis of diversity mechanisms used in evolutionary algorithms. EAs are examined here in a rigorous way using runtime analysis [8,

119, 80]. Previous studies in the field of runtime analysis in the context of diversity have examined how different diversity mechanisms influence the ability of an algorithm to obtain an optimal solution [49, 50]. In this part of the thesis we consider diversity from the decision-space perspective.

This chapter extends the work published in the conference GECCO [53, 54].

The contents of this chapter is organized as follows. In Section 6.2, we introduce the definition of population diversity and the algorithm that is subject of our investigation when considering diversity maximization. Our analysis for the classical *OneMax* problem is presented in Section 6.2 and Section 6.4 shows our results for the *LeadingOnes* problem. The analysis for the two example problems, namely complete bipartite graphs and paths, are discussed in Section 6.5. Finally, we finish this chapter with some concluding remarks to possible topics for future work.

6.2 Background

In this section, some basic ideas of the population diversity maximization of simple optimization problems are introduced.

6.2.1 Decision Space Diversity Measurement

As discussed in Chapter 2, the definition of difference between individuals should be decided first. There are many ways to measure the difference between different individuals and the definition of difference depends on the type of individuals and the main aim of the optimization process.

In this chapter, the subject problems all have pseudo-Boolean functions $f: X \rightarrow \mathbb{R}$ that map elements of the search space $X = \{0, 1\}^n$ to real values. Since pseudo-Boolean functions are defined on bit-strings, we use Hamming distance

$$H(x, y) = \sum_{i=1}^n |x_i - y_i|,$$

where $x = (x_1, \dots, x_n), y = (y_1, \dots, y_n) \in \{0, 1\}^n$, to evaluate the difference between two individuals.

To fulfil the required features mentioned in Chapter 2, which are twinning, monotonicity in varieties and monotonicity in distance, the diversity of a set of solutions P is defined as the sum of Hamming distance between each pair of individuals in P . Note that in general P can be a multi-set which may include duplicates. In order to meet the twinning property, duplicates are removed before computing the diversity of a (multi-)set P based on the Hamming distance.

Algorithm 6.1: $(\mu + 1)$ -EA_D

-
- 1 Initialize P with μ n -bit binary strings.
 - 2 Choose $s \in P$ uniformly at random.
 - 3 Produce s' by flipping each bit of s with probability $1/n$ independently from each other.
 - 4 Check whether s' meets the quality criteria or not. If s' fulfils the quality requirement, then add s' to P and execute $OptDiv(P)$, otherwise go back to step 2.
 - 5 Repeat step 2 to 4 until termination criterion is reached.
-

Algorithm 6.2: Diversity optimization component $OptDiv(P)$

-
- 1 Choose a solution $z \in \{x \in P \mid c(x, P) = \min_{y \in P} c(y, P)\}$ uniformly at random.
 - 2 Set $P := P \setminus \{z\}$.
-

Definition 6.1. For a given population P , the population diversity is defined as

$$D(P) = \sum_{\{x,y\} \in \hat{P} \times \hat{P}} H(x, y),$$

where $\hat{P}$ is the set with all distinct solutions in P .

Moreover, the contribution of solution x is defined as

$$c(x, P) = D(P) - D(P \setminus \{x\}).$$

Implicitly, we define

$$c(x, P) = \begin{cases} 0, & \text{if } \exists y \in P \setminus \{x\} \text{ with } x = y \\ \sum_{y \in \hat{P} \setminus \{x\}} H(x, y), & \text{otherwise.} \end{cases}$$

6.2.2 Population-based Evolutionary Algorithm with Diversity Maximization

Since our aim is to find a set of solutions of different structures, we combine the classical $(\mu + 1)$ -EA with diversity optimization process. The quality criteria of individuals is pre-defined by the decision maker. The $(\mu + 1)$ -EA with solution diversity optimization is defined as $(\mu + 1)$ -EA_D. The whole process of $(\mu + 1)$ -EA_D is given in Algorithm 6.1.

The diversity optimization is conducted until all individuals in the solution set reach the quality requirement. In single-objective problem, the quality is evaluated based on the fitness function. In maximization problem, the quality requirement is expressed as lower bound of the fitness value. Therefore, once entering the diversity optimization process, the algorithm will reject the offspring with fitness below threshold.

If an offspring of acceptable quality is produced, the individual with least contribution to the population diversity is eliminated from the solution set. If this solution is not unique, a solution is chosen uniformly at random among the solutions with the smallest diversity

contribution. Algorithm 6.2 defines the $OptDiv(P)$ component where population diversity gets improved.

6.2.3 Classical Runtime Analysis Method for Evolutionary Algorithms

Although EAs are widely applied to many optimization problems, the theoretical understanding is far behind their practical success due to their randomized behaviours. Knowledge about the behaviours of EAs is of great help in improving them and applying them as well. In the past decades, advances have been achieved in the theoretical analysis of the computational time of EAs. Early stage studies which contribute to the theoretical understanding of EAs focus on classical example functions in single-individual EAs [42, 159] and population-based EAs [70, 162]. In these previous studies, the simplified process is analyzed to find the expected optimization time of the algorithm.

We study our algorithm in terms of the number of fitness evaluations until it has produced a population P with $f(x) \geq v, \forall x \in P$ that has the maximal diversity $D(P)$. We call this the *optimization time* of the algorithm. The *expected optimization time* refers to the expected number of fitness evaluations to reach this goal.

We first analyze the time until all individuals have fitness at least v after having achieved such an individual for the first time. The process is similar to the take-over effect in a population and we show an upper bound of $O(\mu \log \mu)$ for a population of size μ in the following lemma. It will serve later on throughout our analysis.

Lemma 6.1. *Having obtained a population with at least one individual of fitness at least v , the expected runtime until all individuals have fitness at least v is upper bounded by $O(\mu \log \mu)$.*

Proof. Since there is already one individual which has fitness value at least v , one possible method to obtain a population with all individuals fulfil the quality requirement is making duplicates of the best solution until all μ solutions are replaced by the replicas. The probability of making a duplicate of the acceptable solution when there already exist i individuals with fitness value above the threshold in the population is

$$\frac{i}{\mu} \cdot \left(1 - \frac{1}{n}\right)^n = \frac{i}{\mu} \cdot \frac{n-1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{i(n-1)}{e\mu n} \geq \frac{i}{2e\mu}.$$

Before entering the diversity optimization process, we need all of the μ individuals in the population set to have acceptable fitness value. The expected waiting time for this process is at most

$$\sum_{i=1}^{\mu-1} \frac{2e\mu}{i} = 2e\mu \sum_{i=1}^{\mu-1} \frac{1}{i} = O(\mu \log \mu).$$

□

6.3 Diversity Maximization for OneMax Problem

In this section, we investigate the classical *OneMax* problem which has been subject to numerous studies in the area of runtime analysis of evolutionary algorithms [42, 162]. Our goal is to understand how a simple evolutionary algorithm can maximize the diversity of its population for this simple benchmark problem.

6.3.1 OneMax Problem

The OneMax problem is a famous example function. The problem is defined as

$$OneMax(x) = \sum_{i=1}^n x_i.$$

The aim is to maximize the number of 1's in a bitstring.

We first analyze until one solution has fitness at least v . To do this, we follow the ideas of Witt [162] about the analysis of the classical $(\mu + 1)$ -EA.

Let v be the threshold of the fitness value, hence, the acceptable solution should have at least v 1-bits. The diversity optimization process will not begin until all of the solutions in the population have fitness values above the threshold. The maximal fitness value of the current population is denoted by $L = \max_{x \in P} OneMax(x)$. Then we show the upper bound of the time for the algorithm to achieve a solution of fitness at least v for the first time.

Lemma 6.2. *The expected time until $(\mu + 1)$ -EA has obtained a solution x with $OneMax(x) \geq v$ is $O(\mu v + n \log \frac{n}{n-v})$.*

Proof. Let L denote the maximum OneMax value in the current population. One sufficient way to increase L is selecting an individual with fitness value equal to L and flipping one of its 0-bits. Since $(\mu + 1)$ -EA can produce replicas of individuals, for a certain L value, duplicates can be made from the individuals with fitness value L before L improves.

Following Witt's idea [162], we assume that L remains the same before there are $\min\{\frac{n}{n-L}, \mu\}$ duplicates of the individual with fitness L . The expected time for the population to have at least $\frac{n}{n-L}$ duplicates of one of these i individuals with fitness value L is at most

$$\begin{aligned} \sum_{i=1}^{\min\{n/(n-L), \mu\}} \frac{e\mu n}{i(n-1)} &= \frac{e\mu n}{n-1} \sum_{i=1}^{\min\{n/(n-L), \mu\}} \frac{1}{i} \\ &\leq \frac{e\mu n}{n-1} \ln \frac{en}{n-L}. \end{aligned}$$

For a population set which has i individuals with fitness value L , improvement can be made by selecting one of these i individuals and flipping one of its 0-bits. The considered probability is

$$\frac{i}{\mu} \cdot \frac{(n-L)}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{i(n-L)}{e\mu n}.$$

Therefore the expected time for the fitness value to increase is at most $\frac{e\mu n}{i(n-L)}$.

The waiting time of the $(\mu+1)$ -EA achieving the first satisfactory solution equals to the sum of expected waiting time for each L value which includes the time for increasing L and time for duplicating individuals. The expected waiting time for the $(\mu+1)$ -EA getting the first individual with fitness v is at most

$$\sum_{L=0}^{v-1} \frac{e\mu n}{\min\{\mu, n/(n-L)\} \cdot (n-L)} + \frac{e\mu n}{n-1} \sum_{L=0}^{v-1} \ln \frac{en}{n-L}.$$

According to the Harmonic sum,

$$\begin{aligned} \sum_{L=0}^{v-1} \frac{e\mu n}{\min\{\mu(n-L), n\}} &\leq \sum_{L=0}^{v-1} \frac{en}{n-L} + \sum_{L=0}^{v-1} e\mu \\ &\leq en(\ln en - \ln(n-v)) + e\mu v \\ &= en \ln \left(\frac{en}{n-v} \right) + e\mu v. \end{aligned}$$

$$\begin{aligned} \frac{e\mu n}{n-1} \sum_{L=0}^{v-1} \ln \frac{en}{n-L} &= \frac{e\mu n}{n-1} \ln \frac{e^v n^v}{n(n-1)(n-2) \cdots (n-v+1)} \\ &= \frac{e\mu n}{n-1} \ln \frac{e^v n^v (n-v)!}{n!}. \end{aligned}$$

As stated in Stirling's Formula, $e^v n^v < \frac{e^{2v} n^v v!}{v^v \sqrt{2\pi v}}$. We can get

$$\ln \frac{e^v n^v (n-v)!}{n!} < \ln \left(\frac{e^{2v} n^v}{v^v \sqrt{2\pi v}} \cdot \frac{v!(n-v)!}{n!} \right).$$

The Binomial coefficients $\binom{n}{k}$ has the property that

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k}.$$

Hence, we get

$$\begin{aligned}
\frac{e\mu n}{n-1} \sum_{L=0}^{v-1} \ln \frac{en}{n-L} &< \frac{e\mu n}{n-1} \ln \left(\frac{e^{2v} n^v}{v^v \sqrt{2\pi v}} \cdot \frac{v!(n-v)!}{n!} \right) \\
&< \frac{e\mu n}{n-1} \ln \left(\frac{e^{2v} n^v}{v^v \sqrt{2\pi v}} \cdot \left(\frac{v}{n}\right)^v \right) \\
&= \frac{e\mu n}{n-1} \ln \frac{e^{2v}}{\sqrt{2\pi v}} \\
&< \frac{2e\mu nv}{n-1}.
\end{aligned}$$

Thus, the expected waiting time of $(\mu + 1)$ -EA with threshold v is $O(n \log \frac{n}{n-v} + \mu v)$. $\square$

As proved in Lemma 6.1, we already know that after an additional phase of $O(\mu \log \mu)$ all individuals in the population have fitness at least v . The next phase of $(\mu + 1)$ -EA_D is diversity optimization process. The feasible solutions of the problem depend on the value of threshold v , therefore, the following analysis is categorized into two cases based on the threshold.

6.3.2 Analysis of Large Threshold

Firstly, we begin with a simple case where the threshold $v = n - 1$. There are $(n + 1)$ possible solutions which have fitness value above the threshold. The composition of optimal solution set depends on the population size μ .

Theorem 6.1. *Let $v = n - 1$ and $\mu \geq n + 1$, then the expected optimization time of $(\mu + 1)$ -EA_D on OneMax is upper bounded by $O(\mu n + \mu \log \mu + n^2 \log n)$.*

Proof. There are $(n + 1)$ different individuals that have fitness value above the threshold. When $\mu \geq n + 1$, the optimal solution set should contain all of the $(n + 1)$ different individuals. According to our definition of diversity, duplicates will not affect the diversity. Then the $(\mu - n - 1)$ other individuals have no contribution to the diversity.

As stated in Lemma 6.2, when $v = n - 1$, the expected waiting time until $(\mu + 1)$ -EA has obtained a solution with fitness value above the threshold is bounded above by $O(\mu n + n \log n)$.

After the first solution with fitness value above the threshold is produced, the algorithm will focus on producing other individuals with acceptable quality. According to Lemma 6.1, the expected runtime of this procedure is bounded above by $O(\mu \log \mu)$.

We now work under the assumption that all individuals have fitness at least v . Note that $(\mu + 1)$ -EA_D will not accept any solution with fitness value below v . In the worst case, these μ solutions are replicas, so the population diversity equals to 0 at the beginning. The diversity can be improved by producing new solutions from the replicas. Since the duplicates in the population have no contribution to the diversity, they will be replaced by the new individual

which has a higher contribution to the diversity. It does not matter which individual is selected from the population to produce a new solution, since the individual with the least contribution will always be the one to be replaced. If the current population has i different individuals, the probability of creating a new solution with fitness value v is at least

$$\frac{1}{n} \cdot \frac{n-i}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-2} \geq \frac{n-i}{en(n-1)}.$$

The 1^n solution can be produced in any stage by flipping the 0-bit of an individual with fitness value v and will stay in the population. The probability of producing the 1^n solution is

$$\frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en}.$$

Since the duplicated individuals in the population will not affect the population diversity, the duplicates will be all replaced before the optimization finishes. The diversity optimization process will not stop until the optimal diversity is reached, which means the population set contains all possible solutions that fulfil the requirement in fitness value.

The expected time for optimizing the diversity is

$$\begin{aligned} en + \sum_{i=1}^{n-1} \frac{en(n-1)}{n-i} &= en + en(n-1) \sum_{i=1}^{n-1} \frac{1}{i} \\ &\leq en + en(n-1) \ln(en) \\ &< O(n^2 \log n). \end{aligned}$$

Hence, the expected waiting time of the $(\mu+1)$ -EA_D on OneMax with diversity optimization for threshold $(n-1)$ is bounded above by

$$\begin{aligned} &O(\mu n + n \log n) + O(\mu \log \mu) + O(n + n^2 \log n) \\ &= O(\mu n + \mu \log \mu + n^2 \log n). \end{aligned}$$

□

We now study smaller population sizes such that not all possible solutions of fitness at least v can be included in the population. In this case the $(\mu+1)$ -EA_D has to obtain of subset of the $(\mu+1)$ feasible solutions which maximize the population diversity.

Theorem 6.2. *Let $v = n - 1$ and $\mu < n + 1$, then the expected optimization time of $(\mu+1)$ -EA_D on OneMax is upper bounded by $O(\mu n \log(\frac{n}{n-\mu}) + n \log n)$.*

Proof. When $\mu < n + 1$, the population set can not include all possible solutions with fitness value above the threshold. Since the all 1-bit solution only has 1 bit different to other acceptable individuals which have 2 bits different to each other, it will not be in the optimal

solution set. Moreover, every individual with fitness $(n-1)$ has the same Hamming distance to each other, therefore, it does not matter which individual is included in the population set.

The proof for expected time of $(\mu+1)$ -EA achieving the population set with all μ individuals with fitness value above the threshold is the same as that in Theorem 6.1. The expected time is at most $O(\mu n + n \log n + \mu \log \mu)$.

Although the 1^n individual will not be in the optimal population, it is of great possibility that the solution with all 1-bits is introduced in some stage of the diversity optimization process. To increase population diversity, it is sufficient to select the 1^n solution and flip one if its 1-bit in the position where no other individuals in the population have 0-bit in. When the population size is small, the probability of selecting the 1^n solution to produce a new solution is large. Since all the individuals in the population have reached the threshold, the probability of getting the 1^n solution is

$$\frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en}.$$

Then the expected time to produce the 1^n solution is less than $en = O(n)$.

After the 1^n solution is introduced, it will remain in the population until the other individuals all have different patterns. The probability of getting a new solution by flipping one 1-bit of the 1^n individual when there are already i different solutions with fitness value above the threshold is

$$\frac{1}{\mu} \cdot \frac{n-i}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{n-i}{e\mu n}.$$

For the $\mu < n+1$ situation, all of the individuals in the optimal population should be of different structures. Since the contribution of 1^n solution to the population diversity is smaller comparing to those of the individuals with fitness v , the 1^n solution will be replaced by a solution with fitness value v after the other $(\mu-1)$ individuals are different from each other.

Then the waiting time for achieving a population of μ different solutions with fitness $(n-1)$ from the intermediate step is

$$\sum_{i=1}^{\mu-1} \frac{e\mu n}{n-i} = e\mu n \sum_{i=1}^{\mu-1} \frac{1}{n-i} \leq e\mu n (\ln n - \ln(n-\mu)).$$

Having obtained μ individuals of fitness $v = n-1$, the solution 1^n is removed from the population as it has the smallest diversity contribution, and then the optimal population is achieved.

$$\begin{bmatrix}
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0
\end{bmatrix}
\begin{array}{l}
\leftarrow \text{individual 1} \\
\leftarrow \text{individual 2} \\
\leftarrow \text{individual 3} \\
\leftarrow \text{individual 4}
\end{array}$$

FIGURE 6.1: The $\mu \times n$ matrix represents the individuals in a population. In the example, it is a matrix for a population with 4 individuals which are all 8 bits in length. The 7th column is all-1-bit column and the 3rd column is 0-bit column as defined.

Summing up, the expected optimization time is

$$\begin{aligned}
& O(\mu n + n \log n + \mu \log \mu) + O(n) + O(\mu n \log(\frac{n}{n-\mu})) \\
& = O(n \log n + \mu n \log \frac{n}{n-\mu}).
\end{aligned}$$

□

6.3.3 Analysis of Smaller Threshold

We now consider the case where $n/2 \leq v < n-1$ holds. For convenience, we store the population in a $\mu \times n$ matrix where each individual as a row and define the column where there is no 0-bit as *all-1-bit column* and the column where there is only one 0-bit as *0-bit column*. An example is shown in Figure 6.1.

With a smaller threshold v , there exist many feasible solutions with different fitness values. The following lemma shows crucial properties of a population maximizing diversity.

Lemma 6.3. *Let $\mu \leq \binom{n}{v}$. The matrix of a population P represents an optimal population, if the whole matrix contains $\mu(n-v)$ 0-bits and each column contains $\mu(n-v)/n$ 0-bits.*

Proof. There are $\binom{n}{v}$ possible solutions for the *OneMax* problem with threshold v . Assuming $v \geq n/2$ implies that there have to be at least as many 1-bits as 0-bits in each individual. For $\mu \leq \binom{n}{v}$, we show that the optimal population should contains only individuals with fitness value v , since these individuals can make a higher contribution to the overall diversity. Then the total number of 0-bits in the population is $\mu \cdot (n-v)$, w.l.o.g, we assume that $\mu \cdot (n-v)/n$ is an integer.

From the perspective of matrix representing the optimal population, the contribution of each column has no influence on those of other columns, so the population diversity equals to the sum of contribution of every column in the matrix. The contribution of each column should be maximized so that the population diversity is maximized. If there are m 0-bits in a column, the contribution of this column will be $m(\mu - m)$. The population diversity can

be calculated as

$$\sum_{i=1}^n m_i(\mu - m_i),$$

where m_i represents the number of 0-bits in the i th column. The constraint is that the total number of 0-bits in the population is at most $\mu(n - v)$, which can be represented as

$$\sum_{i=1}^n m_i \leq \mu(n - v).$$

Before all columns are balanced in the number of 0-bits, there exist at least two columns that one has more 0-bits than average number and the other has less 0-bits than average number. Let i, j, k represent the number of 0-bits in columns which has 0-bits above, below and equal to the average number separately. Their relationship can be interpreted as $j < k < i$, where $i, j, k \in \mathbb{N}$. Reducing the unbalance rate by flipping a 1-bit and a 0-bit of column with i and j . Increasing j by 1 causes the diversity change by

$$(j + 1)(\mu - j - 1) - j(\mu - j) = \mu - 2j - 1.$$

Decreasing i by 1 causes the diversity change by

$$(i - 1)(\mu - i + 1) - i(\mu - i) = -\mu - 2i - 1.$$

Therefore, the overall change to diversity is

$$(\mu - 2j - 1) + (-\mu - 2i - 1) = 2(i - j - 1).$$

Since i, j and k are all natural numbers and none of them are equal as defined, $2(i - j - 1)$ should be at least 2. Hence, whenever there is unbalance in the number of 0-bits in each column, there exist some columns which can be changed to gain balance and increase diversity, which implies that the population diversity is optimized only when the 0-bits are evenly distributed in each column. The number of 0-bits in each column is then $\mu \cdot (n - v)/n$.

The population diversity reaches the optimality when $m_i = \mu \cdot (n - v)/n$ with the value of

$$n \cdot \frac{\mu v}{n} \cdot \frac{\mu(n - v)}{n} = \mu^2 v(n - v)/n.$$

□

We now consider the case of a small population where $\mu \leq n/(n - v)$ holds. In this case the optimal population contains only individuals which have 0-bits in different positions. The example of global optimum is given in Figure 6.2.

Theorem 6.3. Let $\mu \leq \frac{n}{n-v}$, then the expected optimization time of $(\mu + 1)$ -EA_D on OneMax is upper bounded by $O(\mu n^2 \log \mu)$.

```

0 0 0 1 1 1 1 1 1 1 1
1 1 1 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 0 0 1 0 1
1 1 1 1 1 1 1 0 1 1 0 1 0

```

FIGURE 6.2: The example of global optimum for the case where $\mu \leq \frac{n}{n-v}$.

Proof. According to Lemma 6.1 and 6.2, it takes $O(\mu v + n \log \frac{n}{n-v})$ time to achieve a population with all individuals above the fitness threshold.

Since the population size is $\mu \leq \frac{n}{n-v}$, the population set with optimal diversity value should contain only individuals which have 0-bits in different positions with other individuals. The matrix for the population with optimal diversity value should only have all-1-bit columns and 0-bit columns.

In the worst case, there are $\mu(n-v)$ 0-bits that have duplicates in the same column. In order to achieve the optimal population, the number of columns with more than one 0-bits should be decreased to 0.

At the beginning of the diversity optimization process, the population diversity is 0 as in the worst case where there are only duplicates. The number of all-1-bit columns is v . Before the population diversity reaches the optimal value, there should exist at least one column that has more than one 0-bits. Hence, one way of improving the diversity is selecting an individual with 0-bit not in the 0-bit column and increasing its contribution to diversity. Let the number of 0-bits in i th column be represented by m_i . Then flipping one 0-bit of an individual will cause the contribution change by

$$(m_i - 1)(\mu - m_i + 1) - m_i(\mu - m_i) = -\mu + 2m_i - 1.$$

Flipping one 1-bit in the all-1-bit column will increase the contribution by $(\mu - 1)$. Therefore, flipping a pair of 1-bit and 0-bit as restricted above will change its contribution by

$$(-\mu + 2m_i - 1) + (\mu - 1) = 2(m_i - 1).$$

In order to increase the diversity, the 0-bit chosen should fulfil the condition of $m_i > 1$, which means the 0-bit to be flipped should have duplicates in the same column.

Before the diversity is optimized, there should always exist a column which has more than one 0-bit. We consider the event of selecting an individual which has a 0-bit in the column with $m_i > 1$ and flipping the certain 0-bit together with a 1-bit in one of its all-1-bit columns. According to our analysis above, this event will produce an individual that increases the diversity by at least $2(m_i - 1)$. Let the number of 0-bits with duplicates in the same column be represented by k . Then the probability for such an event described above to happen equals to

$$\frac{k}{\mu} \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-2} \geq \frac{k}{e\mu n^2}.$$

The event described in the last paragraph decreases the number k by 1. When there are two 0-bits in a column, these two duplicated 0-bits can be split into two 0-bit columns in one iteration. Hence, it takes $\mu(n - v) - (n - v)$ steps to get the optimal population.

Therefore the overall waiting time is

$$\begin{aligned} \sum_{k=\mu(n-v)}^{(n-v)} \frac{e\mu n^2}{k} &= e\mu n^2 \sum_{k=\mu(n-v)}^{n-v} \frac{1}{k} \\ &\leq e\mu n^2 (\ln e\mu(n-v) - \ln(n-v)) \\ &\leq e\mu n^2 \ln e\mu \end{aligned}$$

Hence, the expected optimization time is

$$O(\mu v + n \log \frac{n}{n-v}) + O(\mu n^2 \log \mu) = O(\mu n^2 \log \mu).$$

□

In the following, we study how $(\mu + 1)$ -EA_D is able to achieve an optimal population if μ is larger. For a population with maximized diversity when $\mu < \frac{1}{4}n^2$ and $n/2 < v < n$, it should fulfill the following requirements as proved in Lemma 6.3,

- There is no duplicate in the population.
- The number of 0-bits in the population is maximized, which is $\mu(n - v)/2$.
- In the corresponding matrix, the 0-bits are evenly distributed in each column.

We call a population matrix balanced if each column has the same number of 0-bits. Otherwise, we call a population matrix unbalanced.

Lemma 6.4. *Let $\mu < \frac{1}{4}n^2$ and $n/2 < v < n$. If the population matrix is non-optimal and unbalanced then there exists at least one 2-bit flip which strictly improves diversity.*

Proof. In a matrix representing an unbalanced population which does not have optimal population diversity, there must exist two columns in which the number of 0-bits in one column is greater than that of the other as proved in Lemma 6.3.

If there is any duplicate in the population, a new individual is always accepted to replace the duplicate, since a duplicate does not contribute to the population diversity.

When there is no duplicate in the population, let the number of 1-bits in two columns be s_1 and s_2 , where $s_1 > s_2$ and both s_1 and s_2 are integers. The overall contribution of the two columns to the population diversity is $s_1 \cdot (\mu - s_1) + s_2 \cdot (\mu - s_2)$. Since $s_1 > s_2$, there must

exist at least one row where there are 0-bit and 1-bit in corresponding columns. Flipping the certain two bits does not affect the contribution of other columns to the population diversity. Hence, the overall contribution after the event should be

$$(s_1 - 1)(\mu - s_1 + 1) + (s_2 + 1)(\mu - s_2 - 1).$$

Therefore, the change of contribution is $2(s_1 - s_2) - 1$. Since s_1 and s_2 are integer, $s_1 - s_2 \geq 1$. Hence, $2(s_1 - s_2) - 1 \geq 1$. If the offspring is not a duplicate of any existing individuals in the population, this 2-bit flip improves the population diversity by at least 1.

Let the individuals be categorized based on the two bits in the certain columns. Assume d_1, d_2, d_3 denote the number of individuals that have 1-bit and 0-bit, 0-bit and 1-bit and both 1-bits in these two columns. Then we should have $d_1 + d_3 = s_1$ and $d_2 + d_3 = s_2$. Since $s_1 > s_2$, then $d_1 > d_2$. There are more individuals with 1-bit and 0-bit than with 0-bit and 1-bit in the certain two columns. Then there must exist at least one individual that flipping the certain two bit does not produce duplicates.

Therefore, before the population diversity is optimized and the matrix is balanced, there must exist at least one two-bit flip that increases the population diversity. $\square$

Lemma 6.5. *Let $\mu < \frac{1}{4}n^2$ and $n/2 < v < n$. If the population matrix is non-optimal and balanced then there exists a 1-bit or 2-bit flip which strictly improves diversity.*

Proof. When a population has the same number of 0-bits in each column but is still not optimized in population diversity, more 0-bits can be introduced in the population to increase the diversity. There exists at least one individual with number of 0-bits smaller than $(n - v)$. We investigate how $(\mu + 1)$ -EAD can improve the population diversity in this situation and show that there either exists a 1-bit flip on the individual I with less than $(n - v)$ 0-bits or if all 1-bit flips on individual I create duplicates in the population, a 2-bit flip on a Hamming neighbour of individual I is able to increase the population diversity.

If selecting an individual I with less than $(n - v)$ 0-bits and flipping one of its 1-bits does not produce replica of the individuals in the population, the change to population diversity when the offspring replaces its parent can be denoted as $(m + 1)(\mu - m - 1) - m(\mu - m) = \mu - 2m - 1$, where m represents the number of 0-bits in the column where the 1-bit is flipped to 0-bit. Since $m < \mu(n - v)/n < \mu/2$, $\mu - 2m - 1 > 0$. Therefore if the certain 1-bit flip does not produce duplicate, it will increase the population diversity by at least 1.

If such 1-bit flips all generate duplicated individuals, then there should exist a 2-bit flip that increases the population diversity. Since 1-bit flips on individual I always generate duplicates, the population should include all possible individuals which have one more 0-bit than the individual I . By selecting one of these individuals and flipping one of its 1-bits and one of its 0-bits, an offspring not duplicated to any of these individuals can be produced. The new offspring has Hamming distance 3 to the individual I . Replacing individual I with this offspring causes the number of 0-bits increase by 1 in two columns and decrease by 1 in

one column. The population diversity is changed by $2(\mu - 2m - 1) + 2m - \mu - 1 = \mu - 2m - 3$. $m < \lceil \frac{\mu(n-v)}{n} \rceil$. Since m is an integer and $n/2 < v < n - 1$, $m \leq \mu/2 - 2$. Hence $\mu - 2m - 3 > 0$.

The 2-bit flip guarantees the population diversity increases by at least 1 if no duplicate is generated in the process. We show when $\mu < \frac{1}{4}n^2$ and $n/2 < v < n$, there should exist at least one 2-bit flip that does not produce duplicates.

Assume such 2-bit flips create duplicates in all cases and the number of 0-bits in individual I is $L < n - v$. The number of Hamming neighbours of individual I that have one more 0-bits is L . The number of possible individuals generated by a 2-bit flip of the Hamming neighbours of individual I is $\binom{L}{2} \cdot (n - L)$. Supposed that all of the $1 + L + \binom{L}{2} \cdot (n - L)$ individuals are included in the population, the number of 0-bits in each column $m \geq 1 + L + \binom{L}{2} \cdot (n - L - 1)$. Then the total number of 0-bits in the population is $M \geq \left(1 + L + \binom{L}{2} \cdot (n - L - 1)\right) \cdot n$. There exist $M' = \left(L + \binom{L}{2} \cdot (n - L)\right) \cdot (n - L + 1) + (n - L)$ 0-bits in the current population. In order to meet the requirement that all columns have the same number of 0-bits, there should be another $(M - M')$ 0-bits in the population. Since there is threshold for number of 1-bits in all individuals, there should be at least $(M - M')/(n - v)$ more individuals to keep the population balanced.

Hence in the current population there should be individuals of number $\frac{M - M'}{n - v} + 1 + L + \binom{L}{2} \cdot (n - L)$.

$$\begin{aligned}
M - M' &= \left(1 + L + \binom{L}{2} \cdot (n - L - 1)\right) \cdot n - \left(L + \binom{L}{2} \cdot (n - L)\right) \cdot (n - L + 1) - (n - L) \\
&= n(L + 1) + \frac{1}{2}L(L - 1)(n - L - 1)n - \left(L + \frac{1}{2}L(L - 1)(n - L)\right) \cdot (n - L + 1) - (n - L) \\
&= L^2 + \frac{1}{2}L(L - 1)(n(n - L - 1) - (n - L)(n - L + 1)) \\
&= L^2 + \frac{1}{2}L(L - 1)(-n + (L - 1)(n - L)).
\end{aligned}$$

Since $L - 1 \geq v$, $n/2 < v < n - 1$ and $\mu \leq \frac{1}{4}n^2$, the minimum population size should be

$$\begin{aligned}
& \frac{M - M'}{n - v} + 1 + L + \binom{L}{2}(n - L) \\
= & \frac{L^2 + \frac{1}{2}L(L - 1)(-n + (L - 1)(n - L))}{n - v} + 1 + L + \frac{1}{2}L(L - 1)(n - L) \\
= & L \cdot \left(-\frac{(L - 1)n}{2(n - v)} + \frac{1}{2}(n - L)(L - 1) \cdot \frac{L - 1 + n - v}{n - v} + 1 \right) + 1 \\
\geq & L \cdot \left(-\frac{(L - 1)n}{2(n - v)} + \frac{1}{2}(n - L)(L - 1) \cdot \frac{n}{n - v} + 1 \right) + 1 \\
= & L \cdot \left(\frac{(L - 1)n}{2(n - v)} \cdot (n - L - 1) + 1 \right) + 1 \\
> & \frac{1}{4}n^2 + n + 1.
\end{aligned}$$

There is a contradiction with the condition $\mu < \frac{1}{4}n^2$. Therefore, the assumption should be false. There should exist a two-bit flip described as previous that does not produce duplicate.

Then it is proved that if 1-bit flips on individual I produce duplicates, a 2-bit flip should be able to improve it.

Therefore, when $\mu < \frac{1}{4}n^2$ and $n/2 < v < n$, if the population is not optimal in population diversity and balanced, there exists either a 1-bit flip or a 2-bit flip that ensures the improvement in population diversity. $\square$

With Lemma 6.4 and 6.5, it is proved that in a population that is not optimized in population diversity, there always exists at least a 1-bit flip or 2-bit flip that can improve the population diversity.

Theorem 6.4. *Let $\mu < \frac{1}{4}n^2$ and $n/2 < v < n$, then the expected optimization time of $(\mu + 1)$ -EA_D on OneMax is upper bounded by $O(\mu^3nv(n - v))$.*

Proof. The expected time for $(\mu + 1)$ -EA_D obtaining a population set with μ individuals have fitness value above the threshold v is at most $O(\mu n + n \log n + \mu \log \mu)$, as proved in Theorem 6.1.

The maximum population diversity is reached when the number of 0-bits is maximized and these 0-bits are evenly distributed among all columns. The maximum population diversity is

$$n \cdot \frac{\mu(n - v)}{n} \cdot \left(\mu - \frac{\mu(n - v)}{n} \right) = \frac{\mu^2(n - v)v}{n}.$$

The population diversity is increased by at least 1 after each 1-bit flip or 2-bit flip that improves the diversity. The expected runtime of a certain 1-bit flip or 2-bit flip is bounded by $O(\mu n^2)$.

Hence, the expected optimization time of $(\mu + 1)$ -EA_D on OneMax is upper bounded by $O(\mu n + n \log n + \mu \log \mu) + \frac{\mu^2(n - v)v}{n} \cdot O(\mu n^2) = O(\mu^3nv(n - v))$. $\square$

6.4 Diversity Maximization LeadingOnes Problem

In this section we will discuss the expected runtime for $(\mu + 1)$ -EA_D on the classical *LeadingOnes* problem which has been subject to several investigations in the area of runtime analysis [14, 162]. *LeadingOnes* is defined to maximize the length of the uninterrupted sequence of 1-bits starting from the leftmost position of a bitstring.

6.4.1 LeadingOnes Problem

The fitness function of *LeadingOnes* is defined as

$$\text{LeadingOnes}(x) = \sum_{i=1}^n \prod_{j=1}^i x_j,$$

which counts the number of leading 1-bits in a bitstring.

The threshold v represents the minimum number of leftmost 1-bits in the individual. Similar to that of *OneMax* problem, the diversity optimization on *LeadingOnes* problem can also be divided into two stages. The first one is obtaining a population of all individuals with acceptable fitness value and the second one is maximizing the population diversity.

Following the proofs by Witt [162], we demonstrate how $(\mu + 1)$ -EA generates the first solution of fitness value above the threshold for *LeadingOnes*.

Lemma 6.6. *The expected runtime until $(\mu + 1)$ -EA on *LeadingOnes* problem has obtained a solution of fitness value above the threshold v is $O(nv + \mu v \log n)$.*

Proof. Assume L represents the largest number of leading 1-bits among all individuals in the current population and i represents the number of individuals with the fitness value L . For a certain L value, we assume it will not change until there are $\min\{n/\ln(en), \mu\}$ duplicates of the individual with fitness L as stated in Witt [162].

The probability for making a duplicate of the individual with fitness L is at least

$$\frac{i}{\mu} \cdot \left(1 - \frac{1}{n}\right)^n \geq \frac{i(n-1)}{e\mu n}.$$

The expected runtime for making $\min\{n/\ln(en), \mu\}$ duplicates is at most

$$\begin{aligned} \sum_{i=1}^{\min\{n/\ln(en), \mu\}-1} \frac{e\mu n}{i(n-1)} &= \frac{e\mu n}{n-1} \sum_{i=1}^{\min\{n/\ln(en), \mu\}-1} \frac{1}{i} \\ &\leq \frac{e\mu n}{n-1} \ln \frac{en}{\ln(en)} \\ &\leq 2e\mu \ln(en). \end{aligned}$$

After there exist at least $\min\{n/\ln(en), \mu\}$ duplicates, L will be improved by selecting an individual with fitness L and flipping its leftmost 0-bit. The probability for this event to happen is

$$\frac{i}{\mu} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{i}{e\mu n}.$$

Since before the improvement is made, there are already

$$\min\{n/\ln(en), \mu\}$$

replicas, i is equal to $\min\{n/\ln(en), \mu\}$, which makes the expected runtime be

$$\frac{e\mu n}{\min\{n/\ln(en), \mu\}} \leq e\mu \ln(en) + en.$$

The expected runtime of the $(\mu + 1)$ -EA obtaining the first individual with fitness value above the threshold v equals to the sum of waiting time for each L value. Therefore, the overall waiting time is

$$v \cdot \left(2e\mu \ln(en) + \frac{e\mu n}{\min\{n/\ln(en), \mu\}}\right) \leq v(3e\mu \ln(en) + en).$$

In conclusion, the overall waiting time for $(\mu + 1)$ -EA on *LeadingOnes* problem to obtain a solution of fitness value above the threshold v is $O(nv + \mu v \log n)$.

□

6.4.2 Runtime Analysis for $(\mu + 1)$ -EA_D on LeadingOnes

For a *LeadingOnes* problem with threshold v , there are 2^{n-v} different possible solutions. When $\mu > 2^{n-v}$, all of the 2^{n-v} different possible solutions should be contained in the optimal population set and there should be duplicates in the population. According to our definition of diversity, the duplicates will not affect the diversity measurement. The composition of optimal population depends on the population size.

Lemma 6.7. *When $\mu \leq 2^{n-v}$, the optimal population of $(\mu+1)$ -EA_D on LeadingOnes with threshold v has the population diversity $\mu^2(n-v)/4$.*

Proof. Assume that there is a matrix which represents all individuals and each individual as a row, which is similar to the matrix for *OneMax* problem. Let m_i equal to the number of 0-bits in i th column. Then the contribution to diversity of each column can be represented as $m_i(\mu - m_i) = \mu m_i - m_i^2$. In the left v columns, there are only 1-bits so the contribution to diversity is 0. For the following $(n - v)$ columns, when $m_i = \mu/2$, the quadratic function reaches its maximal. The contribution of each column has no effect on those of the other columns. Hence, when there is no duplicate in the population and each of the $(n - v)$

columns has $\mu/2$ 0-bits, the population diversity equals to $(\mu^2/4) \cdot (n - v) = \mu^2(n - v)/4$, which is the maximum value. $\square$

Before the diversity optimization phase starts, $(\mu+1)$ -EA_D works on generating a population with all individuals have requirement quality.

Lemma 6.8. *The expected waiting time of $(\mu + 1)$ -EA_D on LeadingOnes to achieve μ different solutions above the threshold, where $\mu \leq 2^{\frac{n-v}{2}-1}$, is bounded above by $O(nv + \mu v \log n + \mu n \log \mu)$.*

Proof. After the first individual with fitness value above the threshold is achieved in $O(nv + \mu v \log n)$ time, another $(\mu - 1)$ individuals with fitness value above the threshold are produced before the diversity optimization process begins. This process will take $O(\mu \log \mu)$ time as proved in Lemma 6.1.

Since the duplicates make no contribution to the diversity and may interfere the optimization process, we should get rid of the duplicates at the beginning of the diversity optimization process. When there are duplicates in the population, a new individual will always be accepted and replace one of the duplicates. A new individual can be produced by selecting an individual and flipping one bit to become one of its undiscovered Hamming neighbours. An upper bound for the expected number of undiscovered Hamming neighbours of a set of individuals is given in [79] as at least $(n - 2 \cdot r)$ where $0 < |P| \leq 2^r$ and P is the set of discovered individuals. In the *LeadingOnes* problem, the v leftmost bits should be all 1's. Only the $(n - v)$ other bits can be either 0-bit or 1-bit so the expected Hamming neighbours are at least $(n - v - 2 \cdot r)$. Since $\mu \leq 2^{\frac{n-v}{2}-1}$, the expected number fulfils $n - v - 2 \cdot r \geq 2$. Assume the number of non-duplicated individuals in the current population is s . Then the expected number of Hamming neighbours is equal to $(n - v - 2 \log s)$. The probability of obtaining an undiscovered Hamming neighbour is at most

$$(n - v - 2 \log s) \cdot \frac{s}{\mu} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{s(n - v - 2 \log s)}{e\mu n}.$$

Therefore the total time for obtaining a population with μ different individuals is

$$\begin{aligned} \sum_{s=1}^{\mu-1} \frac{e\mu n}{s(n - v - 2 \log s)} &= e\mu n \sum_{s=1}^{\mu-1} \frac{1}{s(n - v - 2 \log s)} \\ &\leq e\mu n \sum_{i=1}^{\mu-1} \frac{1}{s} \\ &\leq e\mu n \ln(e\mu). \end{aligned}$$

Hence, it takes at most $O(\mu n \log \mu)$ time to get a population set with no duplicates in it. Taken all stages into consideration, the expected runtime of $(\mu + 1)$ -EA_D on *LeadingOnes* to

achieve μ different solutions above the threshold is bounded above by

$$\begin{aligned} & O(nv + \mu v \log n) + O(\mu \log \mu) + O(\mu n \log \mu) \\ &= O(nv + \mu v \log n + \mu n \log \mu). \end{aligned}$$

□

Now, we show an upper bound for $(\mu + 1)$ -EA_D on *LeadingOnes* that holds for $\mu \leq 2^{\frac{n-v}{2}-1}$.

Theorem 6.5. *Let $\mu \leq 2^{\frac{n-v}{2}-1}$, then expected optimization time of $(\mu + 1)$ -EA_D on *LeadingOnes* is upper bounded by $O(nv + \mu v \log n + \mu n \log(\mu(n - v)))$.*

Proof. According to Lemma 6.6, it takes at most $O(nv + \mu v \log n + \mu n \log \mu)$ time for $(\mu + 1)$ -EA_D on *LeadingOnes* to get a population of μ different feasible solutions.

After the duplicates are replaced by different solutions, if each column has $\mu/2$ 0-bits, the population diversity should be equal to the maximal value $\mu^2 n/4$ as proved in Lemma 6.7. In the worst case, the initial number of 0-bits m_i in column i is either μ or 0. In order to increase the diversity of the population, m_i should either increase or decrease to $\mu/2$. Since the duplicates have no contribution to the population diversity, in this process, it should be guaranteed that the new individual produced is not a replica of any existing individuals.

Let s_i and t_i represent the number of 0-bits and 1-bits in the i th column respectively. Then $|s_i - t_i| = d_i$ can be regarded as the *unbalance rate* of 0-bits and 1-bits in the i th column. Consider the event that selecting an individual randomly and flipping one of its 0-bits or 1-bits to decrease the unbalance rate of the column. When $s_i < t_i$, this event will cause the population diversity change by

$$(s_i + 1)(\mu - s_i - 1) - s_i(\mu - s_i) = \mu - 2s_i - 1,$$

as the contribution to diversity of the other columns will not change. Since $s_i + t_i = \mu$ and $s_i < t_i$, we get $s_i < \mu/2$. Then the contribution change is at least 0. At the beginning of this stage, there is no duplicate in the population and in each iteration, it should be guaranteed that there is no duplicate introduced to the population. Since there is no duplicate in the parent population, there should exist at most $\min\{t_i, s_i\}$ individuals that are only different in the chosen column from any other individuals. Therefore, there exist at least $|s_i - t_i|$ individuals which have no replicas in pattern without considering the selected column, which also means there should be at least $(s_i - t_i)$ 0-bits that can be flipped without making a duplicate. The probability for such an event to happen is $\frac{1}{\mu} \cdot \frac{1}{n} \cdot (1 - \frac{1}{n})^{n-1} \geq \frac{1}{e\mu n}$.

In a population which is not optimized, there should be $\sum_{i=0}^{n-v} |s_i - t_i|$ different mutations that can lead to diversity improvement through the event we described in the last paragraph. At first, we assume there are either all 1-bits or all 0-bits in each column which makes the total number of feasible mutation equal to $\mu(n - v)$. After each mutation, the number of feasible mutation is decreased by 2 according to the definition of unbalance rate. Let $d = \sum_{i=0}^{n-v} |s_i - t_i|$, then the expected time for the improvement is at most $e\mu n/d$. For the

optimized population, the balance rate of each column should be 0. Before the population diversity is maximized, a 1-bit flip as discussed above causes the unbalance rate of the certain column decreased by 2. Hence, the total waiting time for the population diversity to be maximized is

$$\sum_{d=1}^{\mu(n-v)/2} e\mu n \cdot \frac{1}{d} \leq \frac{1}{2} e\mu n \ln(e\mu(n-v))$$

Hence, the overall runtime of $(\mu + 1)$ -EA_D on *LeadingOnes* is bounded above by $O(n^2 + \mu v \log n + \mu n \log(\mu(n-v)))$. $\square$

6.5 Analysis of Vertex Cover Problem

There are a number of NP-hard combinatorial optimization problems, among which we choose minimum vertex cover as the starting point of population diversity research. Our goal is to understand the diversity optimization process for complete bipartite graphs and single paths. For the vertex cover problem there exist several algorithms that give a 2-approximation of an optimal solution [75, 158] which means they guarantee solutions which is at most twice the size of the optimal cover set. In some cases, the number of 2-approximations might be limited by n , e.g. for the star graph. While, in general, it is hard to determine how many solutions of this quality exist. Therefore, we restrict ourselves to classes of graphs that have many solutions which are 2-approximations and present runtime results for the $(\mu + 1)$ -EA incorporating diversity maximization for complete bipartite graphs and single paths.

In this section, we consider the vertex cover problem which is given by an undirected graph $G = (V, E)$. The goal is to find a minimum set of nodes $V' \subseteq V$ such that each edge is covered, i.e. for each $e \in E$, $e \cap V' \neq \emptyset$. For our investigations, we assume that the considered algorithms start with a population where each individual is already of desired quality. Note that the individuals in the initial population are not required to be all different. Our goal is to analyze the expected runtime until the EAs have obtained a population of good or optimal diversity where all individuals meet the quality criteria.

We also design a simplified version of the $(\mu + 1)$ -EA_D, where a random individual is selected, mutated and then compared to its parent. If the new individual contributes more to the population diversity, it will replace the parent in the population, otherwise it will not be accepted to the population. Define the contribution of x' to the population diversity when replacing an individual x with x' as

$$c_P(x', x) = D(P \setminus \{x\} \cup \{x'\}) - D(P \setminus \{x\}),$$

Algorithm 6.3: $(\mu + 1)$ -EA*_D

- 1 Initialize P with μ n -bit binary strings.
- 2 Choose a solution $s \in P$ uniformly at random.
- 3 Produce s' by flipping each bit of s with probability $1/n$ independently from each other.
- 4 Check whether s' meets the quality criteria or not. If s' does not fulfil the quality requirement, jump to step 2.
- 5 If $c_P(s', s) > c_P(s)$, replace s with s' in P .
- 6 Repeat step 2 to 5 until termination criterion is reached.

1 1 1 1	0 1 1 0 1 0 0 0	-> individual 1
1 1 1 1	0 1 0 0 1 0 1 0	-> individual 2
1 1 1 1	1 0 1 0 1 0 0 0	-> individual 3
	.	
	.	
1 1 1 1	1 1 1 0 0 0 0 1	-> individual $\mu-1$
1 1 1 1	0 0 0 1 1 1 0 0	-> individual μ
set V_1	set V_2	

FIGURE 6.3: The $\mu \times n$ matrix represents the individuals in a population. In the example, each row represent an individual. In each row, the left ϵn bits and right $(1 - \epsilon)n$ represent the existence of nodes from set V_1 and set V_2 in the individual respectively.

where $x \in P$. The termination criterion is defined the same as that in $(\mu + 1)$ -EA_D. The simplified algorithm is defined in Algorithm 6.3 and named as $(\mu + 1)$ -EA*_D. The two algorithms are examined for different problems and different parameters.

The solutions to the vertex cover problem can also be represented as binary string, where each 1-bit denotes the existence of corresponding node in the cover set.

6.5.1 Analysis of Complete Bipartite Graphs

We start by studying the complete bipartite graphs. In a complete bipartite graph, the vertices can be split into two sets V_1 and V_2 , which are of size ϵn and $(1 - \epsilon)n$ respectively. There is an edge between each pair of nodes from set V_1 and V_2 . If the nodes in V_1 is indexed from 0 to $\epsilon n - 1$ and nodes in V_2 is indexed from ϵn to $n - 1$, the cover set can be represented by a binary string with length n . When $\epsilon < \frac{1}{2}$, a cover set consisting of all the nodes in V_1 is the global optimum of the problem. We use matrix to represent the population as shown in Figure 6.3.

In the MVC problem on complete bipartite graph, we focus on the solutions which constitute a 2-approximation of an optimal solution. The population diversity optimization process is conducted on the population after all individuals in the population meet the quality criteria.

The composition of acceptable cover sets depends on the parameter ϵ . Since we focus on the 2-approximation solutions, it is helpful to discuss the different situations based on the different relationship between ϵ , $\frac{1}{2}$ and $\frac{1}{3}$.

6.5.1.1 $\epsilon < 1/3$

Assume $\epsilon < 1/3$. In order to be a 2-approximation of the optimal solution, a cover set should always include every node in set V_1 and at most ϵn other nodes in set V_2 .

In this case, $(\mu + 1)$ -EA_D is investigated. The population is initialized with a 2-approximated solution and $(\mu - 1)$ n-bit binary strings randomly chosen from $\{0, 1\}^n$.

Taken population diversity into consideration, the $(\mu + 1)$ -EA_D aims at finding cover sets of size $2\epsilon n$ and maximizing the population diversity. In order to make sure a solution is 2-approximated to the optimal solution set, the leftmost ϵn bits in the bitstring should be set to 1. Then there are at most ϵn bits need to be selected from set V_2 , which means among the $(1 - \epsilon)n$ bits on the right, there are at most ϵn 1-bits. The diversity optimization process can be seen as a OneMax problem with population size $\mu = (1 - \epsilon)n$ and threshold $v = (1 - 2\epsilon)n$. The analysis follows the ideas about OneMax in Section 6.3.

When $\epsilon < 1/3$,

$$2\epsilon n < (1 - \epsilon)n < \epsilon n.$$

The left ϵn columns representing the set V_1 should all be 1-bits to guarantee the feasibility of an individual. The contribution of the left part to the population diversity is 0. The right $(1 - \epsilon)n$ columns should be balanced in the number of 0-bits and 1-bits in order to achieve the maximum population diversity. The average number of 0-bits in each column is at least $\frac{(1-2\epsilon)n \cdot \mu}{(1-\epsilon)n}$. Since $\epsilon < 1/3$, we get

$$(1 - 2\epsilon)n > \frac{1}{2}(1 - \epsilon)n.$$

Then the average number is greater than $\mu/2$. To maximize the population diversity $\sum_{i=\epsilon n}^n m_i(\mu - m_i)$, where m_i is the number of 0-bits in the i th column, each column should have

$$m_i = \frac{(1 - 2\epsilon)n \cdot \mu}{(1 - \epsilon)n}.$$

Then the optimal population diversity is

$$\frac{(1 - 2\epsilon)\mu n}{(1 - \epsilon)n} \cdot \left(\mu - \frac{(1 - 2\epsilon)\mu n}{(1 - \epsilon)n}\right) \cdot (1 - \epsilon)n = \frac{(1 - 2\epsilon)\epsilon\mu^2 n}{1 - \epsilon}.$$

When there is already one feasible individual in the population, the next stage of diversity optimization is obtaining a population of all individuals satisfying the 2-approximation. Making duplicates of the individual already accepted to fill the population is one possible solution. The probability of making a duplicate of the existing feasible solution to fill up the

population with size μ when there are k feasible solutions in population is

$$\frac{k}{\mu} \cdot \left(1 - \frac{1}{n}\right)^n = \frac{k}{\mu} \cdot \frac{n-1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{k(n-1)}{e\mu n} \geq \frac{k}{2e\mu}.$$

The expected runtime until the whole population is filled up with feasible solutions is upper bounded by

$$\sum_{i=1}^{\mu-1} \frac{2e\mu}{i} = 2e\mu \sum_{i=1}^{\mu-1} \frac{1}{i} = O(\mu \log \mu).$$

After every individual in the population is a 2-approximation to the optimal cover set, the population diversity optimization process begins, according to $(\mu + 1)$ -EA_D. Since when $\epsilon < 1/3$, $\frac{1-\epsilon}{\epsilon} < 2$, in this case we only discuss the situation with the larger population size.

Define P_{ini}^1 as a population with μ individuals among which there is at least one individual that is a 2-approximation to the optimal cover set.

Theorem 6.6. Let $\frac{1-\epsilon}{\epsilon} \leq \mu \leq \frac{1}{9}n^2$ and $\epsilon < 1/3$, then expected optimization time of $(\mu + 1)$ -EA_D on vertex cover problem for complete bipartite graph starting with P_{ini}^1 is upper bounded by $O(\mu^3 n^3)$.

Proof. If population size is at most $\binom{(1-\epsilon)n}{\epsilon n}$, there should be no duplicates in the optimal population. The population should be consist of different 2-approximation solutions which guarantee the balance of 0-bits and 1-bits in each column.

The maximum population diversity is $\frac{(1-2\epsilon)\epsilon\mu^2 n}{1-\epsilon}$. As proved in Lemma 6.4 and 6.5, there is always at least one possible 1-bit flip or 2-bit flip that can improve the population diversity by at least 1. The probability of selecting a certain individual and flipping two bits is

$$\frac{1}{\mu} \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-2} \geq \frac{1}{e\mu n^2}.$$

Therefore, the expected runtime for selecting the certain individual and flipping its two bits is bounded by $O(\mu n^2)$. The overall expected runtime for diversity optimization process equals to

$$O(\mu n^2) \cdot \frac{(1-2\epsilon)\epsilon\mu^2 n}{1-\epsilon} = O(\mu^3 n^3).$$

The expected optimization time for $(\mu + 1)$ -EA_D on MVC for complete bipartite graph starting with P_{ini}^1 is upper bounded by $O(\mu \log \mu) + O(\mu^3 n^3) = O(\mu^3 n^3)$. $\square$

6.5.1.2 $\epsilon = 1/3$

If $\epsilon = 1/3$, a 2-approximation cover set can also be composed of all nodes in the larger set. Then there are two types of possible cover sets fulfil the 2-approximation condition, which are all nodes in set V_1 and at most $\frac{1}{3}n$ nodes in set V_1 , and all nodes in set V_2 . Let A represent the cover set which includes all nodes in V_2 and B_i refer to the cover set with all nodes in set V_1 and at most $\frac{1}{3}n$ in set V_2 , as shown in Figure 6.4.

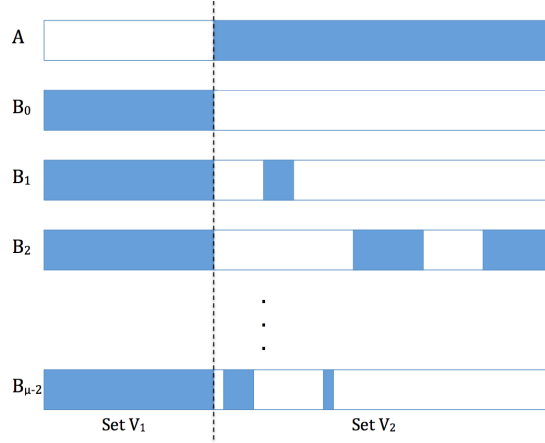


FIGURE 6.4: When $\epsilon = 1/3$, the population with maximized diversity should include cover set with different types. In the graph, the coloured part in each individual represent the nodes selected in the cover set.

In order to maximize the population diversity, A should be included in the population since all of the other solutions have the whole set V_1 selected. From the matrix, the columns representing the existence of nodes in V_1 have either all 1-bits or one 0-bit. Solution A which represents the whole set V_2 makes the contribution of left $\frac{1}{3}n$ columns to the population diversity increase from 0. The average number of 0-bits in each column in the right part of matrix is at least

$$\frac{(\mu - 1)(\frac{1}{3}n)}{\frac{2}{3}n} = \frac{\mu - 1}{2} < \frac{\mu}{2}.$$

Then the number of 0-bits and 1-bits in the right $\frac{2}{3}n$ columns should be equal in order to maximize the population diversity. The population with optimal diversity should have solution A and other $(\mu - 1)$ solutions which have equal number of 0-bits and 1-bits in the right $\frac{2}{3}n$ columns which represent the set V_2 .

The optimal population diversity is

$$\frac{1}{3}n(\mu - 1) + \frac{2}{3}n \cdot \frac{\mu^2}{4}.$$

Define P_{ini}^2 as a population with μ individuals among which there is one solution has all nodes in set V_2 and at least one individual that includes all nodes in set V_1 and at most ϵn other nodes in set V_2 .

Theorem 6.7. Let $\epsilon = 1/3$ and $4 < \mu < \frac{1}{9}n^2$, the expected optimization time of $(\mu + 1)$ - EA_D on vertex cover problem for complete bipartite graph starting with P_{ini}^2 is upper bounded by $O(\mu^3 n^3)$.

Proof. Starting with the population P_{ini}^2 , the left part of the matrix has already reached the optimal situation, which has the contribution of $\frac{1}{3}n(\mu - 1)$ to the population diversity. Balancing the right part of the matrix can be seen as OneMax problem with bitstring length $\frac{2}{3}n$, threshold $\frac{1}{3}n$ and population size $(\mu - 1)$.

There should always exist at least one 2-bit flip that can improve the population diversity as proved in Lemma 1. The optimal population diversity is $(\frac{\mu^2 n}{6} + \frac{(\mu-1)n}{3})$. The 2-bit flip can increase the population diversity from $\frac{(\mu-1)n}{3}$. The probability for a 2-bit flip is $\frac{1}{\mu} \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot (1 - \frac{1}{n})^{n-2} \geq \frac{1}{e\mu n^2}$. Therefore the overall expected runtime is

$$\frac{\mu^2 n}{6} \cdot e\mu n^2 = \frac{1}{6} e\mu^3 n^3 = O(\mu^3 n^3).$$

When $\epsilon = 1/3$ and $4 < \mu < \frac{1}{9}n^2$, the expected optimization time for $(\mu + 1)$ -EAD on vertex cover for complete bipartite graph starting with P_{ini}^2 is upper bounded by $O(\mu \log \mu) + O(\mu^3 n^3) = O(\mu^3 n^3)$. $\square$

6.5.1.3 $1/3 \leq \epsilon < 1/2$

When the size difference between the two sets is smaller, the 2-approximation solution may not include all of the nodes in the optimal solution set. There are two types of acceptable solutions. The acceptable cover set can either be composed by the whole V_1 set and up to ϵn nodes from set V_2 or the whole V_2 set and up to $(2\epsilon n - (1 - \epsilon)n = (3\epsilon - 1)n)$ nodes from set V_1 . Since the numbers of 0-bits and 1-bits should be balanced in order to maximize the population diversity, the optimal solution set should include both types of the acceptable solutions.

If $\mu = 2$, the population with the optimal diversity consists of the two sets. And the maximum population diversity is n .

According to the definition of population diversity, duplicates do not contribute to the diversity. In order to keep the individuals feasible and maximize the population diversity, we need to select the acceptable solution from the two types with less overlap.

When $2 < \mu < 2\epsilon n + 2$, the population with maximum diversity is composed by equal number of solutions in different types. In order to get rid of duplicates, the individuals have all nodes in one set and an extra node in the other set. The extra nodes included are different among the solutions of the same type. The maximum population diversity is

$$\frac{\mu}{2} \cdot \frac{\mu}{2} \cdot (n - \mu + 2) + \left(\frac{\mu}{2} + 1\right) \left(\frac{\mu}{2} - 1\right) (\mu - 2) = \frac{\mu^2 n}{4} - \mu + 2.$$

In the optimal population, the numbers of cover sets of different types are equal. However, equal number of solutions of different type does not guarantee a better population diversity. Let type A and type B represent cover set include all nodes from V_2 and V_1 respectively.

Lemma 6.9. *When $8 - 2\epsilon n < \mu \leq 2\epsilon n + 2$, the population with maximized population diversity contains equal number of solutions of type A and type B .*

Proof. The contribution of each column to the population diversity is independent. The overall population diversity equals to the sum of contribution of all columns. Let M equal

to $\mu^2 n/4$, which is the population diversity of a population where all columns are balanced in the number of 1's and 0's. According to the definition of population diversity, any duplicated individuals do not contribute to the overall population diversity. Since, the solutions to the bipartite graph should contain either set A or set B, there does not exist population with all columns balanced.

When μ is even number, there are three different cases of population composition.

In case 1, there are the same number of individuals in type A and type B in the population. In order to get rid of duplicates but still maximize the population diversity, except for the two individuals that only include set V_1 or set V_2 , the other individuals should include one extra node in the other node set. This causes the contribution of the certain column to the population diversity decreased by 1. Since $\mu \leq 2\epsilon n + 2$, there are enough number of possible solutions to the bipartite graph that fulfil the requirement. The maximum population diversity can be achieved in this case is $M - (\mu - 2)$.

In case 2, there are more individuals of type B than those of type A. The more unbalanced of a column, the less contribution to the population diversity. Therefore, we consider the case where there are 2 more individuals of type B than type A. In this case, it is possible to make the partition representing the existence of node in set V_2 balanced in number of 1's and 0's. For the rest columns, in order to get rid of duplicates, each individual other than the one only includes nodes in set V_2 should include some other nodes in set V_1 . The maximum population diversity can be obtained is $M - \epsilon n - 3(\mu/2 - 2)$ in case 2.

In case 3, we consider that there is 2 more individuals of type A than type B. Similar to case 2, the maximum population diversity is $M - (1 - \epsilon)n - 3(\mu/2 - 2)$, which is always less than the maximum value in case 2.

When $\mu > 8 - 2\epsilon n$, $(\mu - 2) < \epsilon n - 3(\mu/2 - 2)$, then $M - (\mu - 2) > M - \epsilon n - 3(\mu/2 - 2)$.

When μ is odd number, the statement can be proved following the same routine. $\square$

Before the population diversity optimization process begins, assume there are already solutions of both types. Since $1/3 \leq \epsilon < 1/2$, it is possible at some stage of the optimization process, there are all solutions of type A or type B in the population.

As shown in Figure 6.5, in extreme case, mutating the first individual may produce a solution of the other type and the offspring has the higher contribution to the population diversity. After the replacement, there are only solutions of type A in the population. In order to maximize the contribution of the right part to the population diversity, there should be more than one solution of type B. However, when there are k nodes missing in the other set, converting between different types takes $\Omega(\mu n^k)$ time. Therefore, we assume the last solution of a certain type is kept in the population and will not be replaced even the offspring improves the population diversity.

In order to make the process clearer, a modified version of the $(\mu + 1)$ -EA_D is used for this situation. The modified algorithm is defined in Algorithm 6.3.

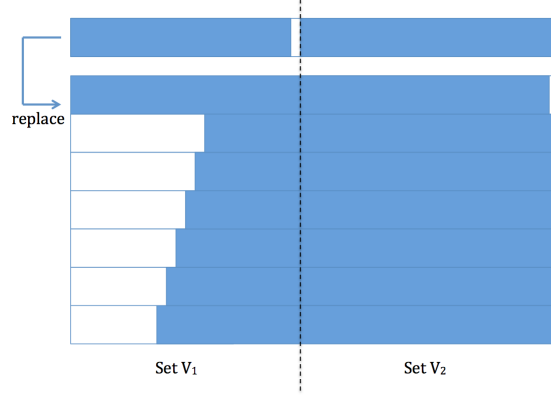


FIGURE 6.5: In the complete bipartite graph with large ϵ , it is possible at some stage, the last individual of one type is replaced by an individual of the other type.

Define P_{ini}^3 as a population with μ individuals in which there is at least one solution of each of type A and B.

Theorem 6.8. *Let $1/3 \leq \epsilon \ll 1/2$ and $8 - 2\epsilon n < \mu \leq 2\epsilon n + 2$, the expected optimization time of $(\mu + 1)$ -EA*_D on vertex cover problem for complete bipartite graph starting with P_{ini}^3 to reach a population with population diversity $(\frac{\mu^2}{4}\epsilon n + (1 - \epsilon)n(\mu - 1))$ is upper bounded by $O(\mu n^2 + \mu^2 n)$.*

Proof. Before the population diversity reaches $(\frac{\mu^2}{4}\epsilon n + (1 - \epsilon)n(\mu - 1))$, there must exist at least one column with number of 1-bits either greater than $\frac{\mu+1}{2}$ or less than $\frac{\mu-1}{2}$. When one individual is selected and each of its n bits is flipped with probability $\frac{1}{n}$, it replaces its parent if the population diversity improves according to $(\mu + 1)$ -EA*_D. Assume during this process, there are m_1 and m_2 columns have 1-bits decreased or increased by 1 respectively. Then the total change of population by the replacement can be represented as

$$2 \sum_{i=0}^{m_1} s_i - m_1(\mu + 1) + m_2(\mu - 1) - 2 \sum_{j=0}^{m_2} s_j.$$

Since for the MVC problem, the solution has to cover either V_1 or V_2 set, after the mutation, the new solution set should also cover either set V_1 or set V_2 . Several bits need to be flipped at the same time to achieve a solution of the other type. With 1-bit flip we can only produce solution that is of the same type of the parent.

Assume in the beginning, there are s solutions of type A in the population. For 1-bit flip, if the offspring replaces its parent, then either $m_1 = 0$ and $m_2 = 1$ or $m_1 = 1$ and $m_2 = 0$. Consider the event that selecting one of the individuals that has 1-bit in the column with more than $\frac{\mu+1}{2}$ 1-bits or 0-bits in the column with less than $\frac{\mu-1}{2}$ 1-bits, flipping the certain bit and replacing its parent. Since the 1-bit flip only influences the certain column, the balance rate of the other column won't change. The total change of the population diversity is either $(2s_i - \mu - 1)$ or $(\mu - 1 - 2s_j)$. Since s_i is selected to be greater than $\frac{\mu+1}{2}$, $2s_i - \mu - 1 > 0$. For s_j less than $\frac{\mu-1}{2}$, $(\mu - 1 - 2s_j)$ is guaranteed to be greater than 0. Although according to the algorithm, the offspring will replace an individual in order to result in the maximum

diversity improvement and this individual may not be the parent if there is other individual contribute less to the population diversity, 1-bit flip can guarantee improvement at least $(2s_i - \mu - 1)$ or $(\mu - 1 - 2s_j)$.

The probability of 1-bit flip is $\frac{1}{\mu} \cdot \frac{1}{n} \cdot (1 - \frac{1}{n})^{n-1} \geq \frac{1}{e\mu n}$. Then the runtime of one 1-bit flip is bounded above by $O(\mu n)$.

1-bit flip won't change the numbers of solutions of type A and type B . As assumption, there is always at least one solution of type A and one solution of type B . Let part M and part N represent the columns in the matrix that represent the existence of nodes in V_1 and V_2 respectively. If there are more solutions of type B in the population, 1-bit flip can guarantee the balance of 1-bits and 0-bits in the part N . In order to contribute more to the population diversity, we need to decrease the number of nodes in V_1 in all the type A solution. We also need to avoid duplicates in the population which do not contribute to the population diversity. During the process, the number of 1-bits in part N either increases or decreases to $\frac{\mu}{2}$. It will take at most $(\frac{\mu}{2} - s)$ steps. For part M , it takes at most $((3\epsilon - 1)n - 1)s$ steps to optimize the diversity. Then there are at most $((\frac{\mu}{2} - s) + ((3\epsilon - 1)n - 1)s = O(\mu + \epsilon n))$ 1-bit flips needed to improve the population diversity to $(\frac{\mu^2}{4}\epsilon n + (1 - \epsilon)n(\mu - 1))$.

Hence, the overall runtime for the 1-bit flip improving the population diversity of vertex cover on complete bipartite graph with $1/3 \leq \epsilon < 1/2$ and $2 < \mu < 2\epsilon n + 2$ to $O(\epsilon\mu^2 n)$ is bounded above by $O(\mu \log \mu) + O(\mu + \epsilon n) \cdot O(\mu n) = O(\mu n^2 + \mu^2 n)$. $\square$

6.5.2 Analysis of Paths

The next sample graph structure we look into is path. The path here refers to a sequence of nodes connected on a path with the vertices on both ends free. The solution to vertex cover problem for paths can also be represented as a binary string where each bit denotes the existence of each node in the cover set. In order to cover every edge, for two nodes next to each other, there should be at least one selected. Therefore there should not be two 0-bits next to each other in the bit strings representing the acceptable cover sets.

Same as the runtime analysis for complete bipartite graph, the aim is to find the optimal solution set with maximum population diversity. In the population diversity optimization process for vertex cover on path, we look into cover set of size that is at most certain threshold v , where v is a predefined number greater than or equal to the size of optimal solution set.

For the path problem, in the beginning of the population diversity optimization process, the population is initialized with μ cover sets which are 2-approximated of the optimum solution. The algorithm investigated in this section is Algorithm 6.1.

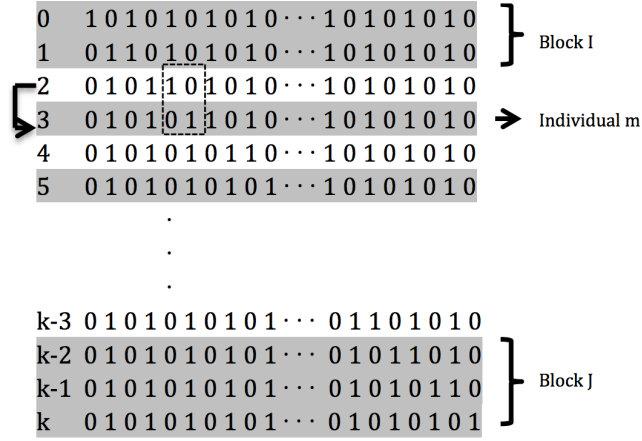


FIGURE 6.6: In set X , all $(k + 1)$ optimal solutions for the path with $n = 2k$ are arranged according to the ascending order of numbers of '01' pairs. The individuals highlighted are the ones in the population.

6.5.2.1 Paths with Even Number of Nodes

Assume the number of nodes $n = 2k$ where k is a positive integer. The optimal matching in this case is not unique. The optimal cover sets have k vertices. The optimal cover set may contain every second node on the path or two nodes next to each other not including the first and last node and every second node from these two. There are $(k + 1)$ different matchings that are optimal as shown in Figure 6.6. At least one of the two nodes connected by an edge should be selected in a cover set in order to cover the certain edge.

Define set X that consists of all optimal solutions of path problem with $n = 2k$, where k is a positive integer. We arrange all the optimal cover sets by ascending the number of '01' pairs and index the solutions from 0 to k . For the i th individual X_i , the j th bit which represents the existence of j th node in the path is defined as x_{ij} , where $0 \leq i \leq k$ and $0 \leq j < n$.

- When $j \leq 2i$, we have $x_{ij} = 0$ if j is even, and $x_{ij} = 1$ if j is odd.
- When $j > 2i$, we have $x_{ij} = 1$ if j is even, and $x_{ij} = 0$ if j is odd.

Figure 6.6 shows the set of optimal solutions and a possible population consisting of optimal solutions.

Theorem 6.9. When $n = 2k$, $\mu \geq k + 1$ and $v = k$, the expected runtime for $(\mu + 1)$ -EA_D producing a population with maximum population diversity is bounded above by $O(\mu n^3)$.

Proof. When the population size $\mu \geq k + 1$, with the threshold $v = k$, the population with maximum diversity should contain all possible optimal solutions and another $(\mu - k - 1)$ duplicates.

According to the definition of the path problem, at least one of the two nodes connected by an edge needs to be selected in the cover set. In the bitstring, there should not be two 0-bits next to each other. In order to get a feasible solution, a pair of 1-bit and 0-bit in an individual

needs to be flipped at the same time. The feasible solution here refers to optimal cover set of the path problem. From each feasible solution, there always exists at least a pair of bits that can be flipped to produce a feasible cover set.

Before all possible solutions are included in the population, there always exists at least one individual that has Hamming distance equal to 2 with the missing solution. The runtime of selecting an individual and flipping its two bits is $O(\mu n^2)$. There are $(k + 1)$ different possible optimal solutions, hence the total expected runtime is $O(\mu n^3)$. $\square$

If the population size is less than the number of all optimal solutions, the feasible solutions are selected in order to maximum the population diversity. In the matrix, the number of 1-bits and 0-bits in each column should be balanced to gain the highest contribution to the population diversity.

Lemma 6.10. *When population size μ is even, the population with maximum diversity is formed by individuals in set X indexed from 0 to $(\frac{\mu}{2} - 1)$ and from $(k - \frac{\mu}{2} + 1)$ to k . When population size μ is odd, the population with maximum diversity is formed by the optimal population with size $(\mu - 1)$ and one more random non-duplicated individual.*

Proof. Let $m \in \mathbb{Z}_{\geq 0}$. From Figure 6.6 it is clear that in $2m$ th column, there are at most $(m + 1)$ 1-bits. There are always the same number of 0-bits in the $(2m + 1)$ th column, which means in the $(2m + 1)$ th column, there are at most $(k - m)$ 1-bits. The contribution of one column to the population diversity is $s_i(\mu - s_i)$, where s_i is the number of 1-bits in the i th column. When $s_i \leq \frac{\mu}{2}$, $s_i(\mu - s_i)$ increases with s_i . Then the highest contribution of the $2m$ th and $(2m + 1)$ th column both are $(m + 1)(k - m)$. When $\mu < k + 1$, the highest contribution of the $2m$ th and $(2m + 1)$ th column in together is $2 \cdot \min\{(m + 1)(k - m), \frac{\mu^2}{4}\}$.

When μ is even, for the first $\frac{\mu}{2}$ cover sets in set X , the number of 1-bits in the $2m$ th column is $\min\{m + 1, \frac{\mu}{2}\}$. In the last $\frac{\mu}{2}$ individuals, the number of 1-bits in the $2m$ th column is $\max\{0, m - k + \frac{\mu}{2}\}$. Considering both of the two blocks, the number of 1-bits in the $2m$ th column should be $\min\{m + 1, \frac{\mu}{2}\}$, when $0 \leq m < \frac{n}{4}$ and $\max\{m - k + \mu, \frac{\mu}{2}\}$, when $\frac{n}{4} \leq m < \frac{n}{2}$. The contribution of each column reaches the highest contribution that is mentioned in the last paragraph. Replacing any of the μ solutions with other individuals causes at least two columns lose balance of 0-bits and 1-bits. Therefore, when μ is even, the population with maximum diversity should be formed by individuals indexed from 0 to $(\frac{\mu}{2} - 1)$ and from $(k - \frac{\mu}{2} + 1)$ to k in set X .

When μ is odd, the population with the individuals in set X indexed from 0 to $(\frac{\mu-1}{2} - 1)$ and from $(k - \frac{\mu-1}{2} + 1)$ to k already has the maximum balance rate in each column. A random non-duplicated individual does not change the balance rate of all columns. Hence, the population with population size $(\mu - 1)$ and maximum diversity together with a random selected non-duplicated solution form the optimal population. $\square$

In Figure 6.6, the i th individual can be achieved by flipping certain two bits of the $(i - 1)$ th or $(i + 1)$ th individual. Assume a population is formed by Block I , Block J and a few other

individuals in between, as shown in Figure 6.6, where Block I includes i individuals indexed from 0 to $(i - 1)$ and Block J includes j individuals indexed from $(k - j + 1)$ to k .

Lemma 6.11. *When $i < j$, replacing the individual k with smallest index among the individuals in between of Block I and J by the i th individual always improves the population diversity.*

Proof. According to our arrangement, in the x th row, the first x pairs of bits should all be '01' and the following $(n - x)$ pairs should be '10'. The i th individual is different from individual k in the $2i$ th to the $2k$ th column. In these columns, there should be '10' pairs for the i th row while all '01' for the individual k .

If the individual k in the population is replaced by the i th individual, then the $2m$ th column has one more 1-bit and the $(2m + 1)$ th column has one less 1-bit, where $i \leq m \leq k$. Let s_i represent the number of 1-bits in the i th column. The change to the population diversity for a pair of columns is

$$(\mu - 2s_{2m} - 1) + (2s_{2m+1} - \mu - 1) = 2(s_{2m+1} - s_{2m} - 1).$$

Assume except for the Block I and Block J , there are p individuals from the partition in between them. Then $s_{2m+1} = j + p$ and $s_{2m} = i$. Hence the change of a pair of columns equals to

$$2(s_{2m+1} - s_{2m} - 1) = 2(j + k - i) > 2(p - 1).$$

Since individual k is one of the p individuals, $p \geq 1$. The change by each pair of columns is non-negative, therefore the total population diversity is improved by the replacement. $\square$

When $j > i$, the individual with the largest index in between can be replaced by the $(k - j)$ th individual in set X , which can be produced by a 2-bit flip of the $(k - j + 1)$ th individual in Block J to improve the population diversity.

Theorem 6.10. *When $n = 2k$ and $\mu < k + 1$, the expected runtime for $(\mu + 1)$ -EA_D producing a population with maximum population diversity for the path problem is bounded above by $O(\mu n^3)$.*

Proof. As proved in Lemma 4, the population diversity can be improved by extending Block I and Block J and making their sizes equal. In order to achieve this, the optimization process needs to start with a population with both of the individuals indexed with 0 and k in set X . In the worst case, there are only duplicates of the $\frac{k}{2}$ th individual. Then it takes at least $(\frac{k}{2} - 1)$ 2-bit flips to achieve the individuals with index 0 or k . The expected runtime of flipping two bits of a certain individual is $O(\mu n^2)$. Then the total waiting time for obtaining these two individuals is bounded above by $2 \cdot (\frac{k}{2} - 1) \cdot O(\mu n^2) = O(\mu n^3)$. Since these two individuals have the highest contribution to the population diversity, they will stay in the population and will not be replaced by any other new solutions.

After these two solutions are in the population, one of the solutions at index $(i - 1)$ in Block I or index $(k - j + 1)$ in Block J is selected and mutated according to the relationship between the size i and j to produce new solution. Then the offspring will replace one of the solutions

lying in between to improve the population diversity. In the worst case, there are $(\mu - 2)$ individuals which need to be replaced by individuals produced by a 2-bit flip of a certain individual. The total runtime is $O(\mu n^2) \cdot (\mu - 2) = O(\mu^2 n^2)$.

Considering both stages, the overall expected runtime of maximizing the population diversity for the path problem with $n = 2k$ and $\mu < k + 1$ is $O(\mu n^3) + O(\mu^2 n^2) = O(\mu n^3)$. $\square$

6.5.2.2 Paths with Odd Number of Nodes

Assume the number of nodes $n = 2k + 1$ where k is a non-negative integer. Then the optimal vertex cover for the path problem contains k nodes. In the optimal cover set, every second nodes are selected along the path from the second node. There exists only one optimal solution for each path problem.

In the path problem, among the two nodes next to each other, there must be at least one selected in the cover set in order to cover the certain edge between them. Therefore in the bitstring a 0-bit needs to be put in between two 1-bits, which means in the bitstring representing a feasible solution, there should not be two 0-bits next to each other. When the threshold $v = k + 1$, there are $\binom{v+1}{k} = \binom{k+2}{k} = \frac{(k+2)(k+1)}{2}$ possible cover sets with size v . Then there are $(1 + \frac{(k+1)(k+2)}{2})$ possible solutions.

Since the duplicates in the population have no contribution to the population diversity according to the definition, when $\mu \geq 1 + \frac{(k+1)(k+2)}{2}$, the population with maximum population diversity should include all possible cover sets.

Theorem 6.11. *When $n = 2k + 1$, $\mu \geq 1 + \frac{(k+1)(k+2)}{2}$ and $v = k + 1$, the expected runtime for $(\mu + 1)$ -EA_D producing a population with maximum population diversity for the path problem is bounded above by $O(\mu n^4)$.*

Proof. Since $\mu \geq 1 + \frac{(k+1)(k+2)}{2}$, any new solution will be accepted and stay in the population.

Let individual a represent the optimal solution '0101...1010' and individual b represent the complement of the optimal solution, which is '10101...0101'.

Assume the statement that before the population diversity is optimized, there is some situation where no existing solution can produce new solution by a 2-bit flip is true.

Before the population covers every feasible solution, if there is no solution with the pattern '110' or '011' in it, then it can only be the solution a or b . By flipping one of the 0-bits of solution a , new solution can be produced. By flipping the leftmost 2 bits or rightmost 2 bits of solution b , we can also get 2 other solutions. These new solutions all have the pattern '110' or '011' in them. If the statement is true, then all of these $(k + 5)$ solutions should already exist in the population.

For the individual with the pattern '110' or '011' in it, new solution can be produced by flipping two bits to form the '101' pattern. If the offsprings are also covered by the population, then the population contains all possible solution sets, which is conflict with our assumption.

Therefore, before all possible solutions are covered in the population, there must exist at least one individual from which flipping one or two certain bits produces a new solution. The expected runtime for producing a new solution is bounded above by $O(\mu n^2)$. When $\mu \geq 1 + \frac{(k+1)(k+2)}{2}$, there are at most $\frac{(k+1)(k+2)}{2}$ new cover sets need to be produced. Then the overall expected runtime for $(\mu + 1)$ -EA_D to produce a population with maximum population diversity for the path problem is bounded above by $O(\mu n^2) \cdot \frac{(k+1)(k+2)}{2} = O(\mu n^4)$. $\square$

6.6 Conclusion

The population of an evolutionary algorithm can be used to reserve a diverse set of solutions where all solutions are of good quality. In this chapter, we examine such approaches in a rigorous way by a first runtime analysis and propose the $(\mu + 1)$ -EA_D which maximizes the diversity of the population once all solutions have reached certain quality criteria.

The analysis is initiated with examination into the population diversity measurement and followed with research into computational complexity of the classical $(\mu+1)$ -EA until achieving a population of solutions satisfying certain quality. As the next step, this chapter is categorized by investigations into different benchmark problems and then subdivided according to different cases of the settings. Our results for the classical benchmark problems *OneMax*, *LeadingOnes* and vertex cover problem on certain graph classes show that the algorithm is efficient in maximizing diversity of the population and keeping quality at the same time.

Our investigations should set the basis for analysis of diversity maximization for classical combinatorial optimization problems and it would be an interesting topic for future work to study the investigated $(\mu + 1)$ -EA_D on other classical combinatorial optimization problems such as the traveling salesperson problem and multi-objective problems.

True wisdom comes to each of us when we realize how little we understand about life, ourselves and the world around us.

Socrates

CHAPTER 7

DIVERSITY MAXIMIZATION FOR MULTI-OBJECTIVE PROBLEMS IN DECISION SPACE

7.1 Introduction

Evolutionary computation has been successfully applied in the area of evolutionary multi-objective optimization [23], such as renewable energy [153] and water network distribution [167]. As introduced in Chapter 2, when using an evolutionary algorithm for solving a given multi-objective problem, the population of the EA is evolved into a set of solutions which represents the trade-off between a set of given objective functions.

Due to the complexity in analysis of population, the theoretical research of evolutionary multi-objective optimization still needs more attention. The key part of an EA for multi-objective optimization is the selection process which decides on which individuals survive for the next generation. Almost all popular selection mechanisms in MOEAs follow the principle of Pareto dominance in an explicit or implicit way. The main difference between MOEAs such as NSGA-II [35], SPEA2 [170] and IBEA [168] is basically the way they differentiate between incomparable solutions.

The decision space diversity in MOEA has become a topic of interests in recent years [145, 155, 156]. The goal is to obtain a set of Pareto optimal solutions which differ according

to the underlying search space. Such set of solutions can be of great interests to decision makers. Having a diverse set of solutions according to the components of a solution gives the decision maker more options of implementing a good solution in different ways. Initial studies on the runtime behaviour for search space diversity optimization have been obtained in [53].

We conduct investigation into the decision space diversity optimization with the classical multi-objective problem OneMinMax in this chapter. Our research contributes to the theoretical understanding of diversity mechanisms in evolutionary multi-objective optimization by means of runtime analysis. This chapter is based on the diversity optimization part of a published paper in conference GECCO [37].

In this chapter, the contents are structured as follows. In Section 7.2, a brief introduction about the problem OneMinMax and the multi-objective EA is included. After that we first present some experimental results about maximizing diversity in OneMinMax and then conduct runtime analysis of the algorithm in Section 7.3. At last, the chapter is concluded in Section 7.4.

7.2 Preliminaries

In the first place, we introduce some basic concepts regarding hypervolume maximization and search space diversity optimization using EAs.

In this chapter, we consider the search space $S = \{0, 1\}^n$, i.e. candidate solutions are bit-strings of length n . In multi-objective optimization, the fitness functions can be considered as a vector value $f : S \rightarrow \mathbb{R}^m$ where $m \geq 2$ is the number of objectives. Assume that all objective functions should be maximized. The fitness of a search point $x \in S$ is given by the vector $f(x) = (f_1(x), \dots, f_m(x))$. We define $f(x) \geq f(x')$ iff $f_i(x) \geq f_i(x')$ for all $i \in \{1, \dots, m\}$. In this case, we say that the objective vector $f(x)$ dominates the objective vector $f(x')$. The set of non-dominated objective vectors is called the Pareto front and the classical goal in multi-objective optimization is to obtain for each objective vector of the Pareto front a corresponding solution. As the Pareto front for most problems is too large, evolutionary multi-objective algorithms evolve a set of solutions that covers the Pareto front in a good way.

7.2.1 OneMinMax Problem

The OneMinMax problem is one of the classical multi-objective problems which are used for theoretical analysis. The problem is defined as,

$$OneMinMax(x) := (\|x\|_0, \|x\|_1),$$

Algorithm 7.1: $(\mu + 1)$ -SIBEA_D

```

1 Start with an initial population  $P$  consisting of  $\mu$  elements from  $S$ .
2 repeat forever
3   Select  $x$  from  $P$  uniformly at random;  $x' \leftarrow \text{mutate}(x)$ ;
4    $\hat{P} \leftarrow P \cup \{x'\}$ ;
5   Let  $z$  be a randomly chosen individual with  $c(z, \hat{P}) = \min_{x \in \hat{P}} c(x, \hat{P})$ ;
6    $P \leftarrow \hat{P} \setminus \{z\}$ ;

```

where the number of 0-bits ($\|x\|_0$) and 1-bits ($\|x\|_1$) in a binary string has to be maximized at the same time. The problem has the property that all search points are on the Pareto front and our goal is to study how evolutionary multi-objective algorithms can obtain diverse sets of solutions with respect to the search and objective space.

7.2.2 Hypervolumn-based Algorithm with Diversity Maximization

The hypervolume indicator is a quality measure of the coverage of a point set which is mapping from the solutions to the objective space of the Pareto-front as introduced in Chapter 2. In particular, given a reference point $r \in \mathbb{R}^m$, the hypervolume indicator is defined in search space S on a set $P \subset S$ as

$$I_H(P) = \lambda \left(\bigcup_{x \in P} [f_1(x), r_1] \times [f_2(x), r_2] \times \cdots \times [f_m(x), r_m] \right),$$

where $\lambda(S)$ denotes the Lebesgue measure of a set S and $[f_1(a), r_1] \times [f_2(a), r_2] \times \cdots \times [f_m(a), r_m]$ is the orthotope with $f(a)$ and r in opposite corners.

We define the contribution of an element $x \in P$ to the hypervolume of a set of elements P as

$$c_H(x, P) = I_H(P \setminus \{x\}) - I_H(P).$$

The algorithm under examination in this chapter is $(\mu + 1)$ -SIBEA which starts with a set P of μ solutions and produces in each iteration from a randomly chosen individual $x \in P$ one offspring x' by mutation resulting in a population $\hat{P} = P \cup \{x'\}$. The mutation operator considered is standard bit mutation which flips each bit of the parent individual x with probability $1/n$. In order to obtain the population in the next generation, an individual $z \in \hat{P}$ with minimal hypervolume contribution is discarded.

To study search space diversity optimization for OneMinMax, we consider a population size that is able to cover the whole Pareto front, i. e. $\mu \geq n + 1$. In this chapter, we analyze $(\mu + 1)$ -SIBEA with a search-space diversity mechanism (see Algorithm 7.1) and study the time until it has produced a population that is diverse with respect to the underlying search space.

There are many ways to measure the difference between different individuals. As discussed in [53], the diversity measurement should have the three properties of twinning, monotonicity in varieties and monotonicity in distance. Since pseudo-Boolean functions are defined on bit-strings, we use Hamming distance

$$H(x, y) = \sum_{i=1}^n |x_i - y_i|,$$

where $x_i, y_i \in \{0, 1\}$, to evaluate the difference between two individuals.

The diversity of a set of solutions P is defined as the sum of Hamming distance between each pair of individuals in P . Note that in general P can be a multi-set which may include duplicates. In order to meet the twinning property [155, 154], duplicates are removed when computing the diversity of a (multi-)set P based on the Hamming distance.

Definition 7.1. For a given population P , the population diversity is defined as

$$D(P) = \sum_{\{x, y\} \in \hat{P} \times \hat{P}} H(x, y),$$

where $\hat{P}$ is the set with all distinct solutions in P .

The diversity optimization is conducted until population covers the whole Pareto-front. The contribution of solution x to the population diversity is defined as

$$c_D(x, P) = D(P) - D(P \setminus \{x\}).$$

Taken both the population diversity and hypervolume indicator into consideration, the contribution of an individual is defined as

$$c(x, P) = (c_H(x, P), c_D(x, P)).$$

For two individuals $x, y \in P$, we define $c(x, P) < c(y, P)$ if $c_H(x, P) < c_H(y, P)$ or $c_H(x, P) = c_H(y, P) \wedge c_D(x, P) < c_D(y, P)$, which indicates y is better than x in quality. And we also define $c(x, P) \leq c(y, P)$ iff $c_H(x, P) \leq c_H(y, P) \wedge c_D(x, P) \leq c_D(y, P)$.

In order to obtain a population which is optimal in both hypervolume indicator and population diversity, we combine the classical $(\mu + 1)$ -SIBEA with the contribution defined above. The $(\mu + 1)$ -SIBEA with solution diversity optimization is defined as $(\mu + 1)$ -SIBEA_D. The whole process of $(\mu + 1)$ -SIBEA_D is given in Algorithm 7.1.

When considering $(\mu + 1)$ -SIBEA_D, we focus on the aspect of maximizing search space diversity. The selecting process involves the hypervolume contribution as the premier component. It has been shown in [120] that $(\mu + 1)$ -SIBEA computes for each Pareto optimal objective vector a corresponding search point, i. e. covers the whole Pareto front, in time

$O(\mu n \log n)$, if $\mu \geq n + 1$. For our investigations regarding search space diversity, we consider population size $\mu = n + 1$. As maximizing the hypervolume is premier goal in $(\mu + 1)$ -SIBEA_D, a population containing for each Pareto optimal objective vector, is obtained in time $O(\mu n \log n)$ following the analysis in [120]. We will work under the assumption that such a population has already been obtained and we are interested in the expected time until such a population has maximal search space diversity.

We study our algorithm in terms of the number of iterations until it has produced a population P that has the optimal hypervolume indicator as well as the maximal diversity $D(P)$. The *expected optimization time* refers to the expected number of fitness evaluations to reach this goal. The population is represented in a $\mu \times n$ matrix where each individual is a row, which allows us to point out when a population has maximal diversity.

7.3 Search Space Diversity Optimization

Assume population size $\mu = n + 1$ and we investigate how evolutionary algorithms can optimize search space diversity under the condition that for each Pareto optimal objective vector at least one search point is contained in the population.

7.3.1 Diversity Maximization for Multi-objective Problem

The following lemma shows crucial properties of a population maximized in population diversity.

Lemma 7.1. *Let $\mu = n + k \leq 2^n$, where $k \geq 1$. If the population P fulfils all of the following properties:*

1. *For each Pareto optimal objective vector v , there is an $s \in P$ with $f(s) = v$.*
2. *There are no duplicated individuals in P .*
3. *Each column of the matrix representing P has either $\lfloor \mu/2 \rfloor$ or $\lceil \mu/2 \rceil$ 1-bits.*

then P is optimal for OneMinMax in population diversity.

Proof. According to the definition of OneMinMax, there are $(n + 1)$ different points in the Pareto-front. Since $\mu \geq n + 1$, the individuals in P have to cover the entire Pareto-front in order to be optimal in the population diversity.

Let P be a population of size μ containing no duplicate and P' be the population obtained from P by replacing at least one of its individuals x by a duplicate of the other $(\mu - 1)$ individuals. According to the monotonicity in varieties property of diversity measurement and Definition 7.1, we have $D(P) > D(P')$ as $\hat{P} = \hat{P}' \cup \{x\}$. This implies that no population containing duplicates can be optimal if $\mu \leq 2^n$.

Let matrix M represent a population P that does not contain any duplicates. We show that P has maximal diversity among all populations containing no duplicates if it contains $\lfloor \mu/2 \rfloor$ or $\lceil \mu/2 \rceil$ 0-bits in each column.

The contribution of each column has no influence on those of any other columns. Hence, the population diversity equals to the sum of the diversity contribution of every column in the matrix. The contribution to population diversity of each column can be written as $m_i(\mu - m_i)$, where m_i represents the number of 1's in the i th column and the overall population diversity of P is given by

$$\sum_{i=1}^n m_i(\mu - m_i).$$

The quadratic continuous function $g(x) = x(\mu - x)$ has the global maximum value of $\mu^2/4$ when $x = \mu/2$. This implies that the maximum is attained for $x = \lfloor \mu/2 \rfloor$ and $x = \lceil \mu/2 \rceil$ when restricting the inputs of g to integers. Hence, P has maximal diversity if it contains $\mu/2$ 1-bits in each column if μ is even. In the case that μ is odd, P has maximal diversity if each column has either $\lfloor \mu/2 \rfloor$ or $\lceil \mu/2 \rceil$ 1-bits.

The maximized population diversity is $\mu^2 n/4$ for μ is even and $(\mu^2 - 1)n/4$ for μ is odd.

□

7.3.2 Experimental Analysis for Diversity Maximization for OneMinMax

One of the possible events that can improve the population diversity is a 1-bit flip in an individual. As the first step, a simple program is implemented to test the possibility of achieving global optimal population in diversity. However, through experimental results, we find out that there are some situation where $(\mu + 1)$ -SIBEA_D is not able to achieve any progress when restricted to 1-bit flips. Lemma 7.1 suggests that the population with maximum diversity and full coverage of the Pareto-front should have balanced number of 1-bits and 0-bits if $\mu = n + 1$ is even. We will see in the proof for $(\mu + 1)$ -SIBEA_D that this is exactly the case. Although 1-bit flip can improve the population diversity in most cases, there are some situations where there does not exist a 1-bit flip that can increase the population diversity.

Some examples are included as Figure 7.1. The populations shown in the example are almost balanced in all columns in numbers of 1-bits and 0-bits but there is no 1-bit flip which can improve the population diversity to optimality. In the first population in Figure 7.1, there are only two columns which are not balanced in the numbers of 0-bits and 1-bits, which are the 1st and 3rd column. Either increasing the number of 1-bits or decreasing the number of 0-bits will improve the population diversity. On the contrary, touching the other columns will decrease the population diversity. Since the offspring after 1-bit flip can only replace the individual with the same objectives in order to keep the coverage of the Pareto-front, the change caused by a 1-bit flip depends on the Hamming distance between the selected individual and its neighbours in the objective space. The change to the population diversity

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	1
1	1	1	1	0	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1
3	4	5	4	4	4	4

0	0	0	0	0	0	0
1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	0	1	1	0	0	0
0	1	1	0	1	1	0
0	0	1	1	1	1	1
0	1	1	1	1	1	1
1	1	1	1	1	1	1
4	4	5	4	4	4	3

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	0	1	1	0
1	1	1	0	1	0	0
1	1	0	1	0	1	1
1	1	1	1	0	1	1
1	1	1	1	1	1	1
4	4	5	4	4	4	3

FIGURE 7.1: The 8×7 matrices represent some populations for which there is no 1-bit flip can improve the population diversity. The last rows report the numbers of 1-bits in corresponding columns.

caused by a 1-bit flip on individual z can be represented as

$$c(z) = S_- - S_+ - \frac{1}{2}(H(z, z') + 1),$$

where S_- and S_+ denote the total number of 1-bits in the columns has one 1-bit decreased and increased respectively. And $H(z, z')$ represents the Hamming distance between the original individual z and the neighbour z' which got replaced by the offspring.

For the example in Figure 7.1, in order to increase the contribution to the population diversity, an offspring should fulfil the requirement of $c(z) > 0$, which means that the columns except for the 1st and 3rd should all remain balanced and the balance rate of these two columns should be increased. It is impossible to improve the population diversity since there is no offspring that is able to increase the contribution of these two certain columns without decreasing the contribution of the other columns.

Flipping the first 0-bit in the all 0 bitstring or the third 1-bit in all 1 bitstring can obtain offspring which has the same contribution to the population diversity which is acceptable by the algorithm. However that event will lead to another population with the same population diversity and there are still these two mutations do not decrease the population diversity, which is the same situation as previous. Therefore no further improvement can be achieved by 1-bit flip for this population.

7.3.3 Runtime Analysis for Diversity Maximization for OneMinMax

Since from experimental results we found exception cases showing that flipping one bit cannot guarantee the population diversity to be maximized, we focus on 2-bit flip to fulfil the task of optimizing population diversity.

Lemma 7.2. *If $\mu = n + 1$ and population diversity is not maximal, then there always exists at least one 2-bit flip in one of the individuals to improve the population diversity.*

Proof. By construction of the Algorithm 7.1, when $\mu = n + 1$, there should exist exactly one individual in the population which refers to each point in the Pareto-front, as proved

in Lemma 7.1. The event that selecting one individual and flipping a 1-bit and a 0-bit of it results in an offspring with the same objective value as its parent. The offspring can only replace its parent and this replacement only happens when the offspring has a larger contribution to the population diversity.

As proved in Lemma 7.1, in a matrix representing a population which does not have optimal population diversity, there must exist two columns that the number of 0-bits in one column is greater than that of the other. Let the number of 0-bits in these two columns be s_1 and s_2 , where $s_1 > s_2$ and both s_1 and s_2 are integers. The overall contribution of the two columns to the population diversity is $s_1 \cdot (\mu - s_1) + s_2 \cdot (\mu - s_2)$. Since $s_1 < s_2$, there must exist at least one row where there is 0-bit and 1-bit in corresponding columns. Flipping the certain two bits does not affect the contribution of other columns to the population diversity. The overall contribution after the event should be

$$(s_1 - 1)(\mu - s_1 + 1) + (s_2 + 1)(\mu - s_2 - 1).$$

Therefore, the change of contribution is $2(s_1 - s_2) - 1$. Since s_1 and s_2 are integers, $s_1 - s_2 \geq 1$. Hence, we get $2(s_1 - s_2) - 1 \geq 1$.

Since the offspring is only compared with its parent, it is impossible to introduce change to the other columns except for these two columns. Therefore, there must exist at least one two-bit flip that should increase the population diversity. $\square$

With this lemma, we can now prove our main result on search space diversity maximization for OneMinMax and show that the $(\mu + 1)$ -SIBEA_D obtains an optimal population in expected time $O(n^3 \log n)$.

Theorem 7.1. *Let $\mu = n + 1$, the expected optimization time of $(\mu + 1)$ -SIBEA_D on OneMinMax is upper bounded by $O(n^3 \log n)$.*

Proof. The algorithm $(\mu + 1)$ -SIBEA obtains a population of maximum hypervolume if $\mu \geq n + 1$ in expected time $O(\mu n \log n)$ as shown in [120]. We assume that a population of maximal hypervolume has already been obtained and investigate how search space diversity is optimized. The Multiplicative Drift Theorem [38] is used to prove the expected runtime bound.

Define $X^{(t)} = D_{OPT} - D(P)$ and $X^{(t+1)} = D_{OPT} - D(P')$, where D_{OPT} denotes the maximum value of the population diversity and P' represents the population in the next generation of P .

Assume at time t there are k 2-bit flips that can improve the population diversity to optimality no matter in what order these k 2-bit flips happen. Such set of events exist for all populations which are not maximized in population diversity if the 2 bits selected is a 1-bit from the columns with more than average number of 1-bits and a 0-bit from the columns with less than average number of 1-bits. The average number of 1-bits refers to $(n + 1)/2$ when n is odd and $n/2$ when n is even. According to Lemma 7.2, such 2-bit flip always

exists before the population diversity is optimized. As long as the certain columns are not balanced in number of 1-bits and 0-bits, the 2-bit flip can improve the population diversity.

According to the algorithm, flipping these certain bits of an individual does not affect the other individuals in the population. The numbers of 1-bits in other columns remain the same except for the two columns, therefore among the other $(k - 1)$ 2-bit flips, the 2-bit flips involving bits in the other $(n - 2)$ columns are still available. Since the k 2-bit flips are selected to improve the population diversity to optimality, if there exists a 2-bit flip involving the two columns, the number of 1-bits of the column should still be unbalanced after the previous event. According to Lemma 7.2, the other 2-bit flips can improve the contribution to population diversity of the columns and then the flips are acceptable. Then the order of the k 2-bit flips does not affect the improvement.

Hence, the k 2-bit flips can be done in any order and result in a population with maximized population diversity as assumption.

The probability for an individual to be selected and two certain bits flipped is

$$\frac{1}{\mu} \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-2} \geq \frac{1}{e\mu n^2}.$$

The probability for one of the k 2-bit flip happen is at least

$$k \cdot \frac{1}{e\mu n^2}.$$

The average expected improvement by the k 2-bit flips is

$$\frac{D_{OPT} - D(P)}{k} = \frac{X^{(t)}}{k}.$$

Then the drift can be represented as

$$E[X^{(t)} - X^{(t+1)}] \geq k \frac{1}{e\mu n^2} \cdot \frac{X^{(t)}}{k} = \frac{X^{(t)}}{e\mu n^2}.$$

In the worst case, the population in the beginning is with the most unbalance rate. It is clear that $s = X^{(t)} \leq D_{OPT}$. The maximum population diversity when $\mu = n + 1$ is

$$\mu \cdot \left(\frac{n+1}{2}\right)^2 = \frac{(n+1)^3}{4}.$$

Therefore we can get $s_0 \leq \frac{(n+1)^3}{4}$ and $s_{min} = 1$.

According to the Theorem 3 in [38], the expected runtime for maximizing the population diversity on OneMinMax is

$$E[T] \leq e\mu n^2(1 + \ln(s_0/s_{min})) = O(n^3 \log n).$$

This completes the proof. □

7.4 Conclusion

Evolutionary multi-objective optimization has been successfully applied to many practical areas in solving real world problems. In this chapter, we have contributed to the theoretical understanding of diversity mechanisms in evolutionary multi-objective optimization by means of rigorous runtime analyses. We have studied a baseline algorithm called $(\mu + 1)$ -SIBEA for the problem OneMinMax.

We integrate the diversity optimization process with the original $(\mu + 1)$ -SIBEA to form another algorithm named $(\mu + 1)$ -SIBEA_D. Through the analysis of experimental results from the implementation of $(\mu + 1)$ -SIBEA_D, some exception cases with bad expected runtime are found. Then we investigate $(\mu + 1)$ -SIBEA in connection with a search space diversity mechanism and show that the algorithm obtains a population of maximal search space diversity covering the whole Pareto front in expected time $O(n^3 \log n)$.

The algorithm $(\mu + 1)$ -SIBEA with diversity mechanism can be applied to other multi-objective optimization problems and our runtime analysis process can be adjusted for other problems as well.

I believe that everything happens for a reason. People change so that you can learn to let go, things go wrong so that you learn to appreciate them when they're right. Sometimes good things fall apart so better things can fall together.

Marilyn Monroe

CHAPTER 8

FEATURE-BASED DIVERSITY OPTIMIZATION FOR PROBLEM INSTANCE CLASSIFICATION

8.1 Introduction

Heuristic search methods including evolutionary algorithms have been shown to be very successful in dealing with various combinatorial optimization problems as discussed in previous chapters. The feature-based analysis of heuristic search algorithms has become an important part in understanding the behaviour of algorithms on different problem instances [109, 146]. This approach characterizes algorithms and their performance for a given problem based on features of problem instances. One of the widely applied feature-based analysis method is based on a set of hard or easy instances constructed by evolving instances using evolutionary algorithms [109, 117, 116]. Although the evolutionary algorithm for constructing such instances is usually run several times to obtain a large set of instances, the researchers still face the problem whether the results in terms of features give a good characterization of problem difficulty.

In this chapter, we propose a new approach of constructing hard and easy instances. Following some recent works on using evolutionary algorithms for generating diverse sets of instances which are all of high quality [154, 155], we introduce an evolutionary algorithm

which maximizes diversity of the obtained instances in terms of a single feature or combination of features. Our approach allows the researchers to generate a set of instances that is much more diverse with respect to the problem feature at hand. The experimental results show that the results from this approach give a much better classification of instances according to their difficulty in being solved by the considered algorithm based on feature values.

To show the benefit of our approach comparing to previous methods, as an example, we examine the classical 2-OPT algorithm for the TSP as introduced in [109]. The experimental results of our new approach show that diversity optimization for the features results in an improved coverage of the feature space over classical instance generation methods. In particular, the results show that for some combinations of two features it is possible to classify hard and easy instances into two clusters with a wider coverage of the feature space comparing to the classical methods. Moreover, the three-feature combinations further improve the classification of hard and easy instances for most of the feature combinations. Furthermore, a classification model is built using these diverse instances that can classify TSP instances based on problem hardness for 2-OPT.

This chapter is based on the results published in the conference PPSN 2016 [52].

This chapter is organized as follows. Firstly, we introduce the known feature list for Euclidean TSP, the designed feature-based diversity measurement and the diversity optimization procedure in Section 8.2. Secondly, the analysis of feature ranges for generated TSP instances is included as Section 8.3. In Section 8.4 and 8.5, a comprehensive investigation into the classification of hard and easy TSP instances for 2-OPT is conducted. Finally, we finish with some concluding remarks in Section 8.7.

8.2 Preliminaries

Our methodology can be applied to any optimization problems and other algorithms, but choosing the Traveling Salesman Problem(TSP) as our subject has the advantage that it has already been investigated extensively from different perspectives including the area of feature-based analysis. Therefore, in this study, we focus on evolving hard and easy instances for the classical NP-hard Euclidean TSP introduced in Chapter 2 .

8.2.1 Traveling Salesman Problem

The input of the problem is given by a set $V = \{v_1, \dots, v_n\}$ of n cities in the Euclidean plane and Euclidean distances $d : V \times V \rightarrow \mathbb{R}_{\geq 0}$ between the cities. The goal is to find a Hamiltonian cycle whose sum of distances is minimal. A candidate solution for the TSP is often

represented by a permutation $\pi = (\pi_1, \dots, \pi_n)$ of the n cities and the goal is to find a permutation π^* which minimizes the tour length given by

$$c(\pi) = d(\pi_n, \pi_1) + \sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1}).$$

For our investigations cities are always in the normalized plane $[0, 1]^2$, i. e. each city has an x - and y -coordinate in the interval $[0, 1]$. In following, a TSP instance always consists of a set of n points in $[0, 1]^2$ and the Euclidean distances between them.

Local search heuristics have been shown to be very successful when dealing with the TSP and the one of the prominent local search operators is the 2-OPT operator [30]. The resulting local search algorithm starts with a random permutation of the cities and repeatedly checks whether removing two edges and reconnecting the two resulting paths by another two edges leads to a shorter tour. If no improvement can be found by carrying out any 2-OPT operation, the tour is called locally optimal and the algorithm terminates.

As in previous studies, we measure hardness of a given instance by the ratio of the solution quality obtained by the considered algorithm and the value of an optimal solution.

The approximation ratio of an algorithm A for a given instance I is defined as

$$\alpha_A(I) = A(I)/OPT(I),$$

where $A(I)$ is value of the solution produced by algorithm A for the given instance I , and $OPT(I)$ is value of an optimal solution for instance I . Within this study, $A(I)$ is the tour length obtained by 2-OPT for a given TSP instance I and $OPT(I)$ is the optimal tour length which we obtain in our experiments by using the exact TSP solver Concorde [4].

8.2.2 Features of TSP Instance

The first issue in the area of feature-based analysis is to identify the features of examined problem and their contribution to the problem hardness. This can be achieved through investigating hard and easy instances of the problem. The structural features are dependent on the underlying problem. In [109], there are 47 features in 8 groups used to provide an understanding of algorithm performance for the TSP, which is discussed in details in Chapter 3. The different feature classes established are distance features, mode features, cluster features, centroid features, MST features, angle features and convex hull features. The feature values are regarded as indicators which allow to predict the performance of a given algorithm on a given instance.

In this chapter, 7 features coming from different feature classes which have shown to be well suited for problem hardness classification and prediction. Instead of the maximum and minimum value of certain feature type, we prefer the mean value. The considered features are:

Algorithm 8.1: $(\mu + \lambda)$ - EA_D

-
- 1 Initialize the population P with μ TSP instances of approximation ratio at least α_h .
 - 2 Let $C \subseteq P$ where $|C| = \lambda$.
 - 3 For each $I \in C$, produce an offspring I' of I by mutation. If $\alpha_A(I') \geq \alpha_h$, add I' to P .
 - 4 While $|P| > \mu$, remove an individual $I = \arg \min_{J \in P} d(J, P)$ uniformly at random.
 - 5 Repeat step 2 to 4 until termination criterion is reached.
-

- *angle_mean* : mean value of the angles made by each point with its two nearest neighbor points
- *centroid_mean_distance_to_centroid* : mean value of the distances from the points to the centroid
- *chull_area* : area covered by the convex hull
- *cluster_10pct_mean_distance_to_centroid* : mean value of the distances to cluster centroids at 10% levels of reachability
- *mst_depth_mean* : mean depth of the minimum spanning tree
- *nnds_mean* : mean distance between nearest neighbours
- *mst_dists_mean* : mean distance of the minimum spanning tree

8.2.3 Evolutionary Algorithm for Evolving Instance with Diversity Optimization

In this research, we introduce our approach of evolving a diverse set of easy or hard instances which are diverse with respect to important problem features.

We propose to use an evolutionary algorithm to construct sets of instances of the TSP that are quantified as either easy or hard in terms of approximation and are diverse with respect to underlying features of the produced problem instances. Our evolutionary algorithm (shown in Algorithm 8.1) evolves instances which are diverse with respect to given features and meet given approximation ratio thresholds.

The algorithm is initialized with a population P consisting of μ TSP instances which have an approximation ratio at least α_h in the case of generating a diverse set of hard instances. In the case of easy instances, we start with a population where all instances have an approximation ratio of at most α_e and only instances of approximation ratio at most α_e can be accepted for the next iteration. In each iteration, $\lambda \leq \mu$ offspring are produced by selecting λ parents and applying mutation to the selected individuals. Offsprings that don't meet the approximation threshold are rejected immediately.

The new parent population is formed by reducing the set consisting of parents and offsprings satisfying the approximation threshold until a set of μ solutions is achieved. This is

done by removing instances one by one based on their contribution to the diversity according to the considered feature.

The core of our algorithm is the selection among individuals meeting the threshold values for the approximation quality according to feature values. The population diversity here is different from the one introduced in Chapter 6 where the difference between individuals is measured based on their structure. In this study, the population diversity is evaluated based on the feature values of each individual. Let $I_1, \dots, I_k$ be the elements of P and $f(I_i)$ be their features values. Furthermore, assume that $f(I_i) \in [0, R]$, i.e. feature values are non-negative and upper bounded by R .

We assume that $f(I_1) \leq f(I_2) \leq \dots \leq f(I_k)$ holds. The diversity contribution of an instance I to a population of instances P is defined as

$$d(I, P) = c(I, P),$$

where $c(I, P)$ is a contribution based on other individuals in the population.

Let I_i be an individual for which $f(I_i) \neq f(I_1)$ and $f(I_i) \neq f(I_k)$. We set

$$c(I_i, P) = (f(I_i) - f(I_{i-1})) \cdot (f(I_{i+1}) - f(I_i))$$

which assigns the diversity contribution of an individual based on the next smaller and next larger feature values. If $f(I_i) = f(I_1)$ or $f(I_i) = f(I_k)$, we set $c(I_i, P) = R^2$ if there is no other individual $I \neq I_i$ in P with $f(I) = f(I_i)$ and $c(I_i, P) = 0$ otherwise. This implies an individual I_i with feature value equal to any other instances in the population gains $c(I_i, P) = 0$.

This diversity measurement is suitable for any features which can be represented by single numerical value. It fulfils the three properties for population diversity measurement mentioned in Chapter 2. Since two different candidate solutions may have the same feature value, a duplicate in this situation may be resulted from duplicated individuals or different individuals. The contribution $c(I, P)$ considers the contribution of the feature value of the individual rather than the individual itself, therefore, it satisfies the requirement of *twinning*. The contribution is calculated based on the difference in feature values between individuals. A distinct feature value has difference more than 0 from the feature values next to it, therefore it has a positive contribution to population diversity.

Furthermore, due to the bonus for non-duplicated extreme feature value, an individual with the unique smallest or largest feature value always stays in the population when working with $\mu \geq 2$.

8.2.4 Experiments Setup

Local search heuristics have been shown to be very successful when dealing with the TSP and the one of the most prominent local search operators is the 2-OPT operator [30]. The resulting local search algorithm starts with a random permutation of the cities and repeatedly checks whether removing two edges and reconnecting the two resulting paths by two other edges leads to a shorter tour. If no improvement can be found by carrying out any 2-OPT operations, the tour is called locally optimal and the algorithm terminates. In this study, we use the implementation introduced in [109].

We carry out our diversity optimization approach for the 7 features discussed in last section and use the evolutionary algorithm to evolve for each feature a diverse population of instances that meets the approximation criteria for hard or easy instances given by the approximation ratio thresholds.

All programs in our experiments are written in R and run in R environment [130]. The functions in `tspmeta` package is used to compute the corresponding feature values [109].

The setting of the evolutionary algorithm for diversity optimization used in our experiments is as follows. We choose $\mu = 30$ and $\lambda = 5$ for the parent and offspring population size, respectively. According to the approximation ratio $\alpha_A(I)$, the instances are categorised into hard and easy. The 2-OPT algorithm is executed on each instance I five times with different initial solutions and we set $A(I)$ to the average tour length obtained. The examined instance sizes n are 25, 50 and 100, which are denoted by the number of cities in one instance. Based on previous investigations in [109] and initial experimental investigations, we set $\alpha_e = 1$ for instances of size 25 and 50, and $\alpha_e = 1.03$ for instances of size 100. Evolving hard instances, we use $\alpha_h = 1.15, 1.18, 1.2$ for instances of size $n = 25, 50, 100$, respectively.

The mutation operator picks in each step one city from the given parent instance uniformly at random and changes its x - and y -coordinator by adding an offset according to the Normal-distribution with standard deviation σ . Coordinates that are out of the interval are reset to the value of the parent. Based on initial experiments we use two mutation operators with different values of σ . We use $\sigma = 0.025$ with probability 0.9 and $\sigma = 0.05$ with probability 0.1 in a mutation step. The evolutionary algorithm terminates after 10,000 generations which allows to obtain a good diversity for the considered features. For each $n = 25, 50, 100$ and each of the 7 features, a set of easy and hard instances are generated, which results in 42 independent runs of the $(\mu + \lambda)$ -EAD.

8.3 Range of Feature Values

Firstly, the diversity optimization approach is evaluated in terms of the diversity that is obtained with respect to a single feature. Focusing on a single feature in each run provides the insight of the possible range of a certain feature value for hard or easy instances. The

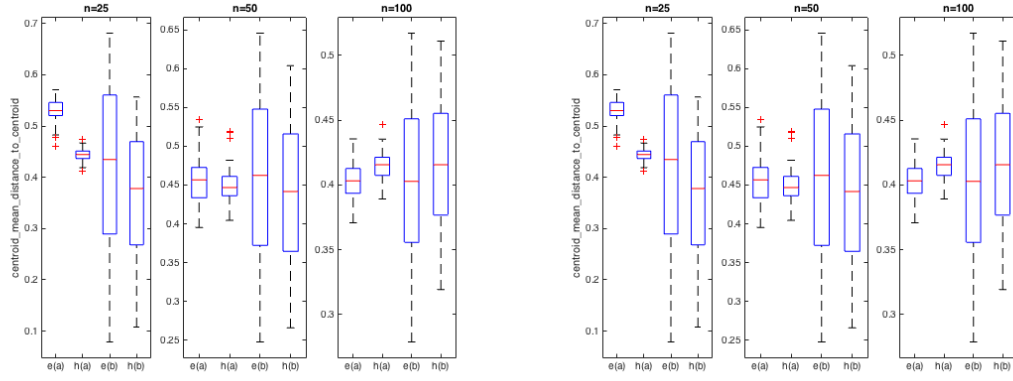


FIGURE 8.1: (left)The boxplots for centroid mean distance to centroid feature values of a population consisting of 100 different hard or easy TSP instances of different number of cities without or with diversity mechanism.(right)The boxplots for cluster 10% distance distance to centroid feature values of a population consisting of 100 different hard or easy TSP instances of different number of cities without or with diversity mechanism. Easy and hard instances from conventional approach and diversity optimization are indicated by e(a), h(a) and e(b), h(b) respectively.

previous study [109] suggests that there are some differences in the possible range of feature values for easy and hard instances. We study the effect of the diversity optimization on the range of features by comparing the instances generated by diversity optimization approach to the instances generated by the conventional approach in [109]. Evolving hard instances based on the conventional evolutionary algorithm, the obtained instances have mean approximation ratios of 1.12 for $n = 25$, 1.16 for $n = 50$, and 1.18 for $n = 100$. For easy instances, the mean approximation ratios are 1 for $n = 25, 50$ and 1.03 for $n = 100$. Hence the instances from two approaches are with similar quality measured by approximation ratio.

Figure 8.1 (left) presents the variation of the mean distance of the distances between points and the centroid feature (*centroid_mean_distance_to_centroid*) for hard and easy instances of the three considered sizes 25, 50 and 100. Each set consists of 100 instances generated by independent runs as mentioned in [109]. As shown in Figure 8.1 (left) the hard instances have higher feature values comparing to easy instances for all instance sizes. For example, in the case of instance size 100, for the hard instances the median value which is indicated by the red line is 0.4157 while it is only 0.4032 for the easy instances. The respective range of the feature value is 0.0577 for the hard instances and 0.0645 for the easy instances. For the instances generated by diversity optimization approach (easy and hard instances are indicated by e (b) and h (b) in Figure 8.1 respectively), there is a difference in the median feature values between the hard and easy instances similar to the instances generated by the conventional approach. Additionally, the range of the feature values for both the hard and easy instances has significantly increased. For example, for the instance size 100, the median value for easy instances is 0.4028 and the range is 0.2382. For the hard instances of the same size, the median is 0.4157 while the range is 0.1917 (see Figure 8.1 (left)).

Similarly, Figure 8.1 (right) presents the variation of cluster 10% distance to centroid (*cluster_10pct_distance_to_centroid*) feature for the hard and easy instances generated by the conventional approach (indicated by e(a) and h(a)) for the hard and easy instances generated by diversity optimization (indicated by e(b) and h(b)). The general observations from these box plots are quite similar to those from the *mst_dist_mean* shown in Figure 8.1 (left). For the easy instances of size 100, the range of the feature value is 0.0919 for conventional instances and 0.3471 for the instances generated by diversity optimization. Similarly, for the hard instances the range of the feature values has increased from 0.0577 to 0.1776 by the diversity optimization approach. As shown in Figure 8.1 (right), there is a significant increase in the range for other instance sizes as well. Improved ranges in feature values are observed for all considered features, however, due to space limitations these are not included in the paper.

The above results suggest that the diversity optimization approach has resulted in a significant increase in the coverage over the feature space. Having the threshold for approximation ratios (α_e and α_h) set, our method guarantees the hardness of the instances. These approximation thresholds are more extreme than the mean approximation values obtained by the conventional method. Furthermore, starting with initial population of duplicated instances and a hard coded threshold, the modified $(\mu + \lambda)$ -EA is able to achieve hard instances with approximation ratio 1.347, 1.493 and 1.259, respectively for instance size 25, 50 and 100. The majority of the instances are clustered in a small region in the feature space while some other points are dispersed across the whole space. This is evident in the median values similar to the values for the instances with respect to conventional approach and with significantly larger range in feature value. The conventional approach has failed to explore certain regions in the feature space and missed some instances existing in those regions. Being able to discover all these instances spread in the whole feature space, our approach provides a strong basis for more effective feature based prediction.

As a result of the increased ranges and the similar gap in median feature values for hard and easy instances compared to the conventional instances, there is a strong overlap in the ranges of the features for easy and hard instances generated by the diversity optimization. This is observed in the results for *mst_dist_mean* and *cluster_10pct_distance_to_centroid* shown in Figure 8.1. Similar pattern holds for the other features as well. This prevents a good classification of problem instances based on single feature value.

In Figure 8.2, some sample hard TSP instances for 2-OPT are shown for different instance sizes with the corresponding optimal tours computed by Concorde. As the diversity of feature value increases, it is possible to generate harder instances and the instance shapes become more diverse. The main observations can be summarized as follows:

- The instance shapes for the smaller instance size structurally differ from those of larger instance sizes. The structure of large hard instances is more complicated than that of smaller instance.
- There is no fixed shape of the whole instance with certain hardness. However, an instance involves one or more U-shape structures as part of its optimal tour is likely to

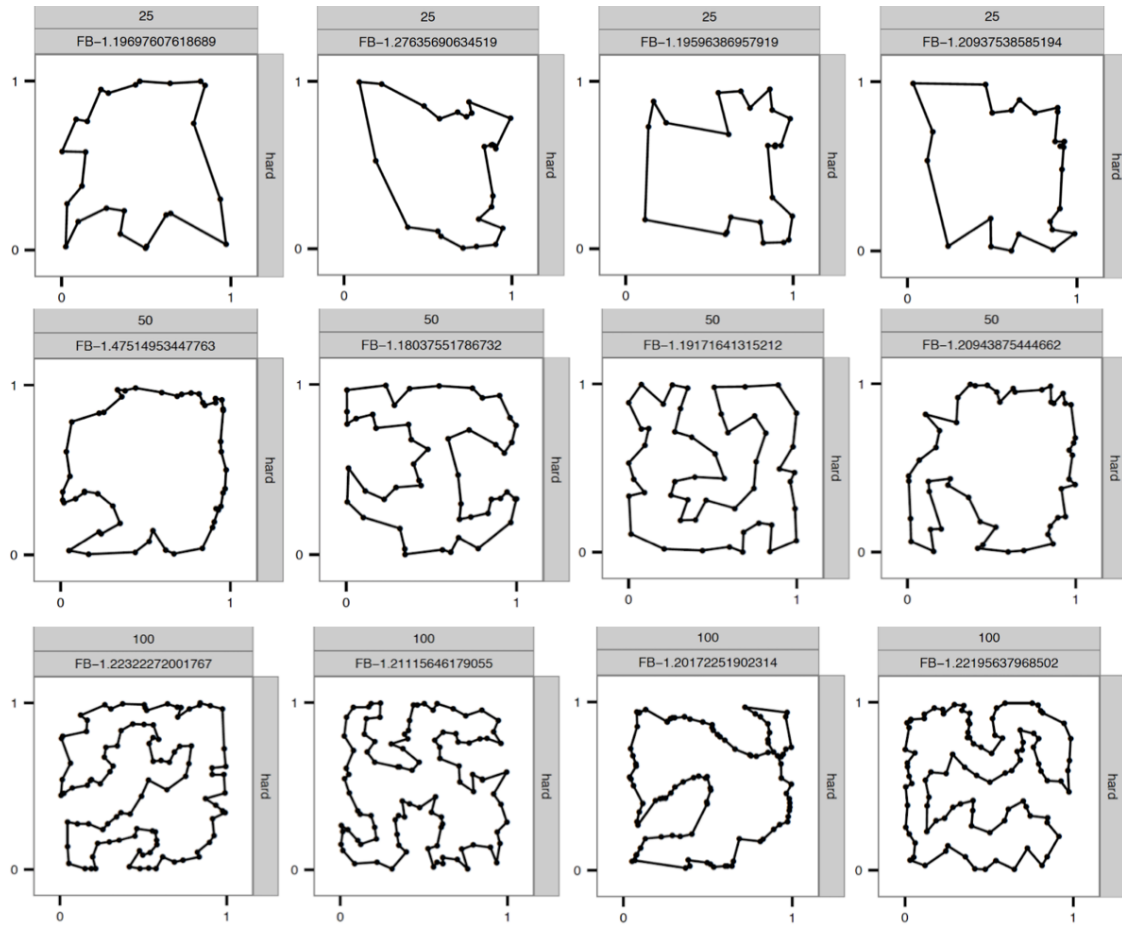


FIGURE 8.2: Some examples of the evolved hard TSP instances of different number of cities are shown with an optimal tour computed by Concorde. The approximation ratio for 2-OPT of each instance example is included after the label 'FB'.

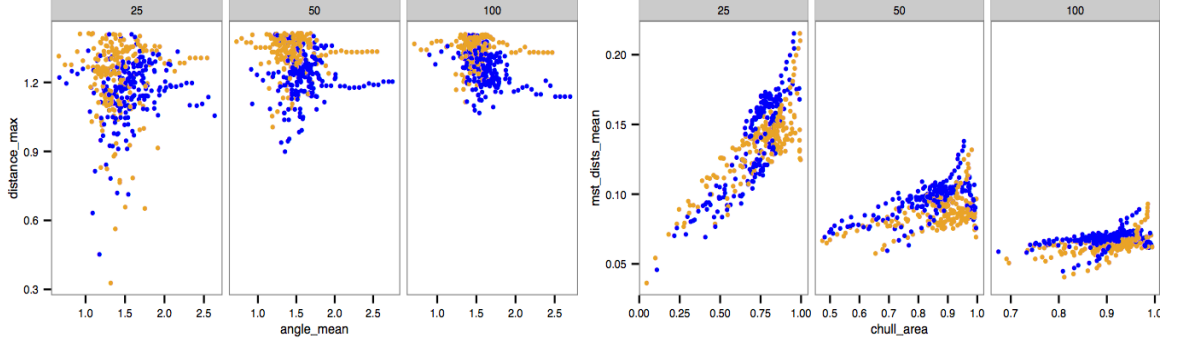


FIGURE 8.3: 2D Plots of feature combinations which provide a separation between easy and hard instances. The blue dots and orange dots represent hard and easy instances respectively.

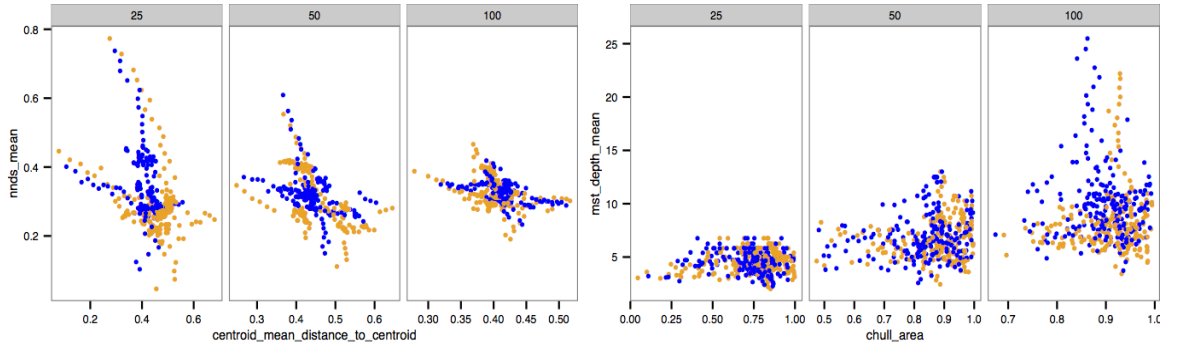


FIGURE 8.4: 2D Plots of feature combinations which does not provide a clear separation between easy and hard instances. The blue dots and orange dots represent hard and easy instances respectively.

have higher approximation ratio, especially for large instance sizes.

8.4 Instance Classification Based on Multiple Features

As a single feature is not capable in clearly classifying the hard/easy instances, combinations of two or three different features are examined in the following. Any two features of the 47 features can be selected and plotted in the 2D space of the data set. Our analysis mainly focuses on combinations of the 7 previously introduced features.

8.4.1 Diversity Maximization over Single Feature Value

Firstly, we represent the instances according to the combination of two different features by points in the 2-dimensional feature value space (see Figure 8.3 for an example).

According to the observation and discussion in [109], the two features *distance_max* and *angle_mean* can be considered together to provide an accurate classification of the hard and easy instances. Whereas after increasing the diversity over the seven different feature values and a wider coverage of the 2D space is achieved, the separation of easy and hard instances

is not so obvious. The clusters of dots representing hard and easy instances have some overlapping as shown in the left figure of Figure 8.3. There are large overlapping areas lying between the two groups of instances. Another example of some separation given by two-feature combination is *mst_dists_mean* and *chull_area* which measure the mean distance of the minimum spanning tree and the area of the convex hull. However, as the number of cities in an instance increases, the overlapping area becomes larger. It is hard to do classification based on this.

After examining the 21 different combinations of two features out of the seven features, we found out that some combinations of two features provide a fair separation between hard and easy instances after increasing the diversity over different feature values. As shown in Figure 8.3, taking both *mst_dists_mean* and *chull_area* features into consideration, some separations can be spotted between hard and easy instances. However, most combinations are not able to give a clear classification between hard and easy instances, for example in Figure 8.4, neither the combination of features *nnds_mean* and *centroid_mean_distance_to_centroid* nor features *mst_depth_mean* and *chull_area* shows clear classification between instances of different hardness. Moreover, along with the instance size increment, the overlapping area of the dots standing for hard and easy instances grows.

Since the majority of two-feature combinations are not capable of classifying easy and hard instances, the idea of combining three different feature is put forward. As in the analysis of two-feature combination, the values of the three selected features are plotted in 3D space.

By considering a third feature in the combination, in the 35 different combinations, it is clear that there are some separations between the two groups of 210 instances from the 3D-plots. A good selection of features results in an accurate classification of the instances. The three-feature combinations with the features measuring statistics about minimum spanning tree always provide good separation between hard and easy instances as shown in Figure 8.5 and Figure 8.6. Although there is an overlapping in the area between the two clusters of hard and easy instances, from the 3D-plots, we can spot some areas where there are only dots for instances of certain hardness.

Taken another feature value into consideration, the two-feature combination that is not able to provide good separation can give some clear classification in hard and easy instances. An example illustrating this is included as Figure 8.7, where together with an additional feature *mst_dists_mean*, the two-feature combination of features *mst_depth_mean* and *chull_area* shows a clear separation between easy and hard instances comparing to the results shown in the left graph in Figure 8.4.

From the investigation of both the two-feature combination and three-feature combination, we found out that the range of feature values for larger TSP instances is smaller. Some of the good combinations for classifying the hardness of smaller instances may not work for larger instances, such as centroid features which perform well when combining with another feature in classifying the hardness of instances of 25 cities while do not show a clear separation with instance size 50 and 100 in our study. Nevertheless, there exist some

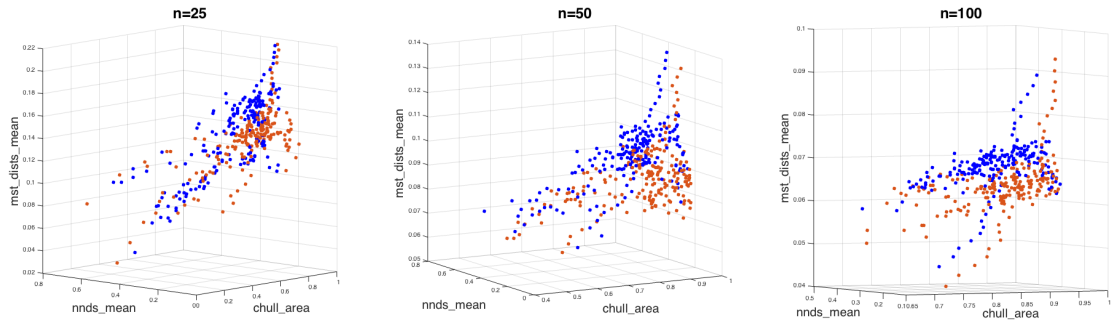


FIGURE 8.5: 3D Plots of combining experiment results from maximizing the diversity over features *mst_dists_mean*, *nnds_mean* and *chull_area*, which provides a good separation of easy and hard instances. Hard and easy instances are represented as blue dots and orange dots respectively.

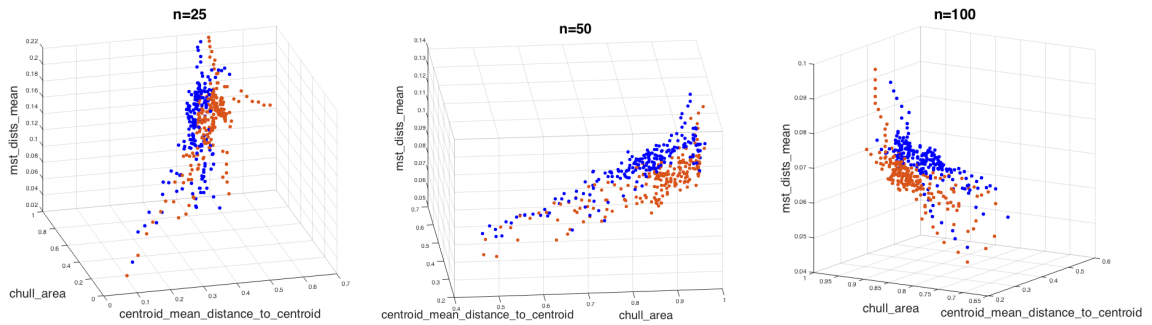


FIGURE 8.6: 3D Plots of combining experiment results from maximizing the diversity over features *mst_dists_mean*, *chull_area* and *centroid_mean_distance_to_centroid*, which provides a good separation of easy and hard instances. Legend is the same as that in Figure 8.5.

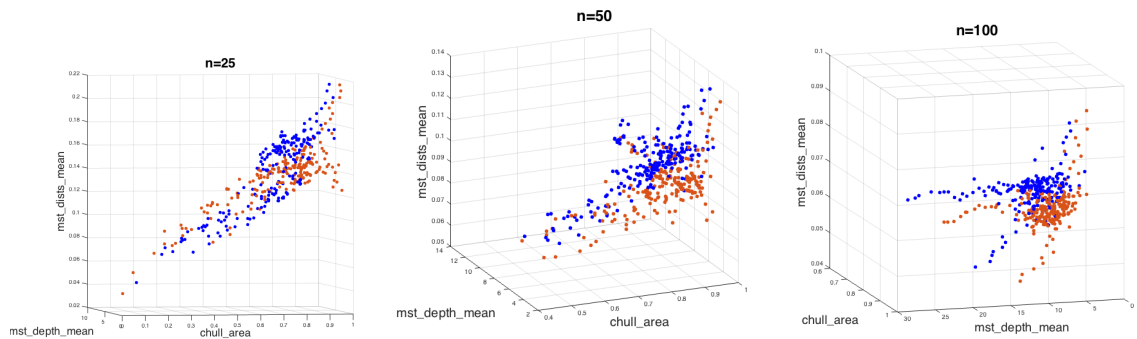


FIGURE 8.7: 3D Plots of combining experiment results from maximizing the diversity over features *mst_dists_mean*, *chull_area* and *mst_depth_mean*, which provides a good separation of easy and hard instances. Legend is the same as that in Figure 8.5.

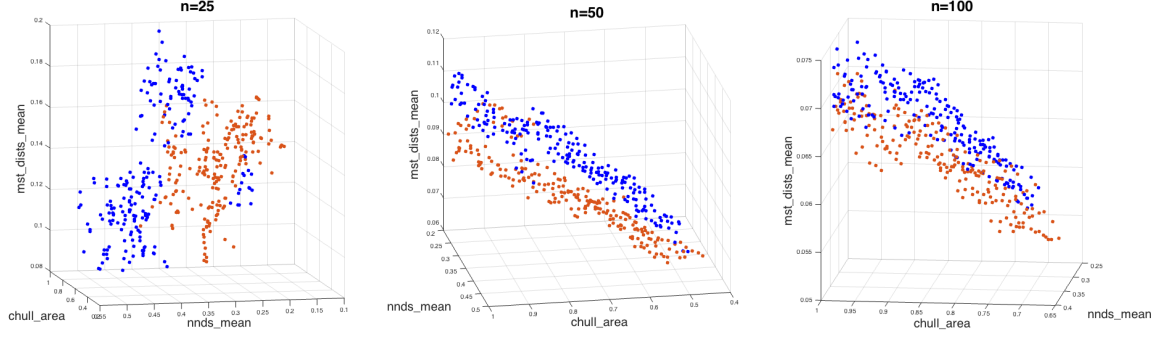


FIGURE 8.8: 3D Plots of combining experiment results from maximizing the diversity over features mst_dists_mean , $nnds_mean$ and $chull_area$ with considering of weighting, which provides a good separation of easy and hard instances. Legend is the same as that in Figure 8.5.

three-feature combinations that give good classification of easy and hard instances without regarding to the instance size, for example mst_dists_mean , $chull_area$ and $nnds_mean$, and mst_dists_mean , $chull_area$ and mst_depth_mean .

8.4.2 Diversity Maximization over Multiple Feature Values

In order to examine the relationship between feature combination and hardness of the instances, a weighted population diversity based on multiple features is introduced. The weighted population diversity for a certain set of features $\{f_1, f_2, \dots, f_k\}$ is defined as the weighted sum of the normalised population diversity over these k features. The contribution of an instance I to the weighted population diversity is defined as

$$d'(I, P) = \sum_{i=1}^k (w_i \cdot d_{f_i}(I, P)),$$

where $d_{f_i}(I, P)$ denotes the normalised contribution to the population diversity $d(I, P)$ over certain feature i and w_i represents the weight of feature i . The contribution of an individual to the population diversity on certain feature is normalised based on the maximum population diversity on the feature, in order to reduce the bias among different features.

This weighted population diversity is used in Algorithm 8.1 to gain some insight of the relationship between features combination and instance quality. The same parent and offspring population sizes are used for these experiments, which are $\mu = 30$ and $\lambda = 5$. The instance sizes examined are still 25, 50 and 100. The 2-OPT algorithm is executed five times to obtain the approximation quality. In the experiments, $(\mu + 1)$ -EA_D execute for 10,000 generation as previous. Since it is shown in Section 8.4 that a combination of three features is able to provide a good separation between hard and easy instances, some of the good three-feature combinations are chosen for exploration. The weight distributions for $\{f_1, f_2, f_3\}$ considered in the experiments are $\{1, 1, 1\}$, $\{2, 1, 1\}$, $\{1, 2, 1\}$, $\{1, 1, 2\}$, $\{2, 2, 1\}$, $\{2, 1, 2\}$, $\{1, 2, 2\}$. The same hardness thresholds are used in these experiments as previous. After the seven

independent runs for easy and hard instances, the final solution sets are put together. Therefore the results set has 210 instances for each instance size and hardness type, which is the same as previous experiments. The results are plotted in 3D space and compared to the previous experiments on single feature discussed in Section 8.3 and 8.4.

The weighted population diversity offers a way to examine the overlapping area of hard and easy instances. With the weighting technique, it takes consideration about the relationship between the different features examined. Since most of these features are not independent from each others and the weighted population diversity considers multiple features at the same time, it is predictable that with the weighted population diversity the extreme value for each single feature may not reach.

An example is shown in Figure 8.8 focusing on maximizing the weighted population diversity over the combination of features *mst_dists_mean*, *nnds_mean* and *chull_area*, which is shown to be a good combination for separating the hard and easy instances. From the comparison between Figure 8.5 and Figure 8.8, we can see that although the results from maximizing weighted population diversity does not cover a wider search space, it provides a detailed insight into the intersection between the hard and easy instances. The 3D plots of different instance sizes show that the combination of these three certain features provide a clear separation between hard and easy instances. There are some overlapping areas in the search space, but it is clear that this combination of features provide some hints for predicting of hard or easy instances.

8.5 Instance Classification Using Support Vector Machine

Support vector machines (SVMs) are well-known supervised learning models in machine learning which can be used for classification, regression and outliers detection [27, 64]. In order to quantify the separation between instances of different hardness based on the feature values, SVM models are constructed for each combination of features.

8.5.1 SVM with Linear Kernel

The linear classifier is the first model tried in classifying the dataset. In SVM the linear classifiers that can separate the data with maximum margin is termed as the optimal separating hyper-plane. From the plots in Figure 8.3, 8.4, 8.5 and 8.6, it is clear that none of the datasets are linearly separable. Taken the trade-off between maximizing the margin and minimizing the number of misclassified data points into consideration, the soft-margin SVM is used for classification.

Let ACC_n be the training accuracy of a feature combination in separating the hard and easy instances of size n . We define ACC_n as the ratio of number of instances which are correctly classified by the model to the total number of instances in the dataset. All classification experiments are done in R with library{e1071} [110]. The training data of the SVM models

are the population of 420 instances generated as in Section 8.3 and the training accuracy is regarded as a quantified measurement of the separation between hard and easy instances. The feature combinations used for classification are the 21 two-feature combinations and 35 three-feature combinations discussed in Section 8.4.

From experiment results, ACC_{25} for two-feature combinations lie in the range of 0.5095 to 0.7548 with an average accuracy of 0.6672, while the ACC_{25} for three-feature combination lie between 0.6286 to 0.7786 with average value 0.7079. In the case of instances with city number of 50, two-feature combination results in ACC_{50} lying in the range of 0.5286 to 0.7738 with an average of 0.6544 while ACC_{50} of three-feature combinations are from 0.5381 to 0.85 with average accuracy equal to 0.6969. For larger instance size, ACC_{100} are in the range between 0.5738 and 0.8119 with average 0.6986 for two-feature combination, whereas those for three-feature combination lie in the scope of 0.6238 to 0.8524 with average 0.7382.

Although three-feature combinations show better accuracy in separation of hard and easy instances than those two-feature combinations, there is no significant difference in ACC for two-feature combinations and three-feature combinations. Moreover, the general low accuracy implies the high possibility that the linear models are not suitable for separating the hard and easy instances based on most of the feature combinations.

We then move to applying kernel function for non-linear mapping of the feature combination.

8.5.2 Nonlinear Classification with RBF Kernel

The linearly non-separable features can become linearly separable after mapped to a higher dimension feature space. The Radial Basis Function (RBF) kernel is one of the well-known kernel function used in SVM classification.

There are two parameters need to be selected when applying RBF, which are C (cost) and γ . The parameter setting for RBF is crucial, since increasing C and γ leads to accurate separation of the training data but at the same time causes over-fitting. The SVMs here are generated for quantifying the separation rate between hard and easy instances rather than classifying other instances. After some initial trials, (C, γ) is set to $(100, 2)$ in all the tests to avoid over-fitting. This parameter setting may not be the best parameters for the certain feature combination in SVM classifying, but it helps us to gain some understanding of the separation of hard and easy instances generated from previous experiments based on the same condition.

Table 8.1 and 8.2 show the accuracy of different two features or three features combination in hard and easy instances separation. With RBF kernel, SVM with certain parameter setting can generate a model separating the dataset with average accuracy of 0.8170, 0.8244 and 0.8346 in 2D feature space for instance size 25, 50 and 100 respectively. Whereas with three features, SVM with the same parameter setting provides a separation with average accuracy of 0.9503, 0.9584 and 0.9422 for instance size 25, 50 and 100 respectively.

Feature 1	Feature 2	ACC_{25}	ACC_{50}	ACC_{100}
angle_mean	centroid_mean_distance_to_centroid	0.8476	0.8071	0.8071
angle_mean	chull_area	0.7857	0.7810	0.7929
angle_mean	cluster_10pct_mean_distance_to_centroid	0.7810	0.7786	0.8000
angle_mean	mst_depth_mean	0.7524	0.7381	0.8000
angle_mean	nnds_mean	0.8167	0.8833	0.8452
angle_mean	mst_dists_mean	0.8119	0.8024	0.8405
centroid_mean_distance_to_centroid	chull_area	0.8619	0.7667	0.8381
centroid_mean_distance_to_centroid	cluster_10pct_mean_distance_to_centroid	0.8524	0.8357	0.7548
centroid_mean_distance_to_centroid	mst_depth_mean	0.8381	0.7643	0.8095
centroid_mean_distance_to_centroid	nnds_mean	0.8786	0.9524	0.8476
centroid_mean_distance_to_centroid	mst_dists_mean	0.8905	0.8571	0.8762
chull_area	cluster_10pct_mean_distance_to_centroid	0.8000	0.7881	0.8548
chull_area	mst_depth_mean	0.7429	0.7429	0.7571
chull_area	nnds_mean	0.8071	0.8905	0.8452
chull_area	mst_dists_mean	0.8619	0.8643	0.9024
cluster_10pct_mean_distance_to_centroid	mst_depth_mean	0.7619	0.7714	0.7929
cluster_10pct_mean_distance_to_centroid	nnds_mean	0.8190	0.8833	0.8643
cluster_10pct_mean_distance_to_centroid	mst_dists_mean	0.8095	0.8095	0.8738
mst_depth_mean	nnds_mean	0.7786	0.8595	0.8405
mst_depth_mean	mst_dists_mean	0.8095	0.8214	0.8810
nnds_mean	mst_dists_mean	0.8500	0.9143	0.9024

TABLE 8.1: This table lists the accuracy of SVM with RBF kernel separating the hard and easy instances in 21 different two-feature space.

Feature 1	Feature 2	Feature 3	ACC_{25}	ACC_{50}	ACC_{100}
angle_mean	centroid_mean_distance_to_centroid	chull_area	0.9500	0.9190	0.9452
angle_mean	centroid_mean_distance_to_centroid	cluster_10pct_mean_distance_to_centroid	0.9405	0.9357	0.8214
angle_mean	centroid_mean_distance_to_centroid	mst_depth_mean	0.9548	0.9548	0.9214
angle_mean	centroid_mean_distance_to_centroid	nnds_mean	0.9452	0.9952	0.9833
angle_mean	centroid_mean_distance_to_centroid	mst_dists_mean	0.9571	0.9500	0.9524
angle_mean	chull_area	cluster_10pct_mean_distance_to_centroid	0.9524	0.9310	0.8881
angle_mean	chull_area	mst_depth_mean	0.9357	0.9238	0.9500
angle_mean	chull_area	nnds_mean	0.9405	0.9714	0.9571
angle_mean	chull_area	mst_dists_mean	0.9667	0.9619	0.9143
angle_mean	cluster_10pct_mean_distance_to_centroid	mst_depth_mean	0.9214	0.9143	0.9810
angle_mean	cluster_10pct_mean_distance_to_centroid	nnds_mean	0.9476	0.9690	0.9333
angle_mean	cluster_10pct_mean_distance_to_centroid	mst_dists_mean	0.9571	0.9143	0.9405
angle_mean	mst_depth_mean	nnds_mean	0.9310	0.9762	0.9238
angle_mean	mst_depth_mean	mst_dists_mean	0.9476	0.9262	0.9476
angle_mean	nnds_mean	mst_dists_mean	0.9429	0.9762	0.8833
centroid_mean_distance_to_centroid	chull_area	cluster_10pct_mean_distance_to_centroid	0.9476	0.9333	0.9310
centroid_mean_distance_to_centroid	chull_area	mst_depth_mean	0.9595	0.8762	0.9762
centroid_mean_distance_to_centroid	chull_area	nnds_mean	0.9667	0.9881	0.9929
centroid_mean_distance_to_centroid	chull_area	mst_dists_mean	0.9714	0.9714	0.8381
centroid_mean_distance_to_centroid	cluster_10pct_mean_distance_to_centroid	mst_depth_mean	0.9476	0.9286	0.8571
centroid_mean_distance_to_centroid	cluster_10pct_mean_distance_to_centroid	nnds_mean	0.9643	0.9905	0.8810
centroid_mean_distance_to_centroid	cluster_10pct_mean_distance_to_centroid	mst_dists_mean	0.9500	0.9595	0.9190
centroid_mean_distance_to_centroid	mst_depth_mean	nnds_mean	0.9500	0.9881	0.9595
centroid_mean_distance_to_centroid	mst_depth_mean	mst_dists_mean	0.9548	0.9548	0.9595
centroid_mean_distance_to_centroid	nnds_mean	mst_dists_mean	0.9667	1.0000	0.9952
chull_area	cluster_10pct_mean_distance_to_centroid	mst_depth_mean	0.9286	0.9524	0.9333
chull_area	cluster_10pct_mean_distance_to_centroid	nnds_mean	0.9524	0.9667	0.9667
chull_area	cluster_10pct_mean_distance_to_centroid	mst_dists_mean	0.9595	0.9595	0.9929
chull_area	mst_depth_mean	nnds_mean	0.9381	0.9857	0.9476
chull_area	mst_depth_mean	mst_dists_mean	0.9476	0.9738	0.9833
chull_area	nnds_mean	mst_dists_mean	0.9714	0.9857	0.9667
cluster_10pct_mean_distance_to_centroid	mst_depth_mean	nnds_mean	0.9214	0.9857	0.9738
cluster_10pct_mean_distance_to_centroid	mst_depth_mean	mst_dists_mean	0.9500	0.9476	0.9643
cluster_10pct_mean_distance_to_centroid	nnds_mean	mst_dists_mean	0.9643	0.9833	0.9976
mst_depth_mean	nnds_mean	mst_dists_mean	0.9429	0.9929	0.9929

TABLE 8.2: This table lists the accuracy of SVM with RBF kernel separating the hard and easy instances in 35 different three-feature space.

Algorithm 8.2: $(\mu + \lambda)$ - EA_{DA}

-
- 1 Initialize the population P with μ TSP instances of certain size.
 - 2 Let $C \subseteq P$ where $|C| = \lambda$.
 - 3 For each $I \in C$, produce an offspring I' of I by mutation and add I' to P .
 - 4 While $|P| > \mu$, remove an individual $I = \arg \min_{J \in P} d_{ar}(J, P)$ uniformly at random.
 - 5 Repeat step 2 to 4 until termination criterion is reached.
-

From the results, it can be concluded that there are better separations between hard and easy instances in the 3D feature space.

8.6 Diversity optimization for Instance Hardness

In the experiments in previous sections, the main focus of diversity maximization is the feature values. The quality of solutions is guaranteed by a predefined threshold. In order to have an insight into the relationship between the feature values and problem hardness, we conduct another experiment with population diversity optimization with instance hardness. The experiment is based on the TSP and Algorithm 8.1 as well. In this case, the approximation ratio is taken as a feature value. Maximizing the diversity over approximation ratio results in a set of individuals with different quality. By doing this, we obtain knowledge about the relationship between feature values and instance hardness from another point of view.

In this case, the algorithm needs to be modified in the solution quality check process. The evolutionary algorithm without hard coded solution quality threshold is shown in Algorithm 8.2. The population diversity measurement $d_{ar}(I, P)$ in step 4 follows the formulation proposed in Section 8.2.3 with the approximation ratio as the target feature value. In the survivor selection phase, the individual that contributes the least to the population diversity over approximation ratio is removed.

For the purpose of reasonable coverage over the whole space, the population size and offspring population size is set to 400 and 10 in this experiment. The other parameters are all kept the same with previous experiments. Then the R program is run for 30,000 generation to obtain some stable results.

Figure 8.9 and 8.10 show some example 3D plots of the experimental results. The 400 individuals are plotted in the 3D feature space with consideration over different feature combinations. The vertical colorbar lying in the right of each plot displays the relationship between color and problem hardness and indicates the mapping of approximation ratio into the different color. The dots in lighter color imply easier instances.

The plots in Figure 8.9 present the resulting instances in the space of feature *chull_area*, *mst_dist_mean* and *mst_depth_mean*. There is clear separation between the red dots and yellow dots in the figure for different problem size. The red and yellow dots refer to instances with extreme approximation ratio in each case. For larger instances, there is no clear clusters

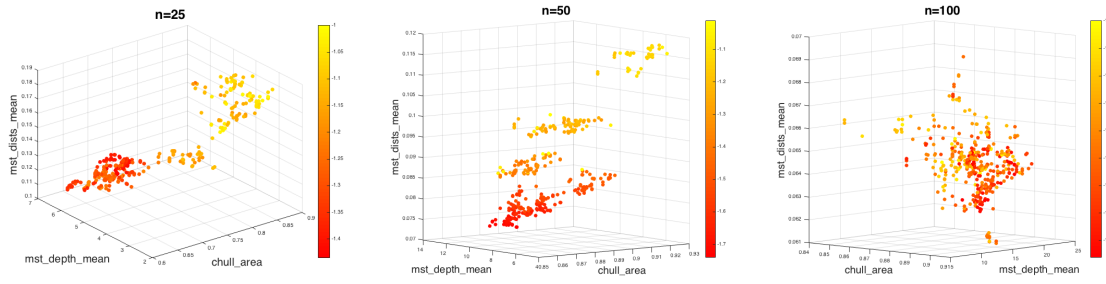


FIGURE 8.9: 3D Plots of experiment results from maximizing the diversity over approximation ratio in the feature space of feature combination *chull_area*, *mst_dist_mean* and *mst_depth_mean*. The color of the dot reflect the hardness of the problem.

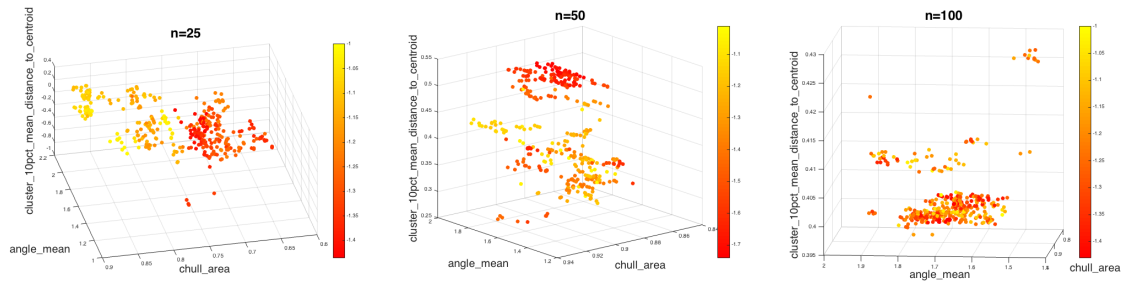


FIGURE 8.10: 3D Plots of experiment results from maximizing the diversity over approximation ratio in the feature space of feature combination *chull_area*, *angle_mean* and *cluster_10pct_mean_distance_to_centroid*. The color of the dot reflect the hardness of the problem.

of instances with different hardness but we can still find areas where hard or easy instances gather. In the other hand, for problem instances with approximation ration lying in the range of 1.1 to 1.2, 1.2 to 1.4 and 1.15 to 1.2 which are represented by orange dots in the plots, the locations are hard to classify.

The plots in Figure 8.10 show the relationship of problem hardness with the feature combination *chull_area*, *angle_mean* and *cluster_10pct_mean_distance_to_centroid*. Similar observation can be obtained. The separation for problem size 25 is clear. However, when the number of cities in each instance increases, the overlapping area becomes larger.

Algorithm 8.2 allows us to generate a set of solutions with different hardness. The approximation ratio for problem size 25, 50 and 100 is maximized to 1.434, 1.739 and 1.430 in this experiment which implies the populations cover a even wider range than those from previous experiments.

8.7 Conclusion

Investigating heuristic search algorithms with respect to features of the underlying problems has become very popular in recent years. In this chapter, we have introduced a new

methodology of evolving easy/hard instances which are diverse with respect to feature sets of the optimization problem at hand.

Using the proposed diversity optimization approach we have shown that the easy and hard instances obtained by our approach cover a much wider range in the feature space than previous methods. The diversity optimization approach provides instances which are diverse with respect to the investigated features. The proposed population diversity measurement provides good evaluation of the variety over single or multiple feature values.

Since different combinations have shown different performance in classifying instances based on hardness, one possible approach for future work is to apply multi-objective algorithm for diversity optimization on multiple feature values.

Our experimental investigations for 2-OPT and TSP have shown that our large set of diverse instances can be classified quite well into easy and hard instances when considering a suitable combination of multiple features which provide some guidance for predication as the next step. This approach can be generalized to other problems with appropriate features. In particular, the SVM classification model built with the diverse instances that can classify TSP instances based on problem hardness provides a strong basis for future performance prediction models which lead to automatic algorithm selection and configuration. Building such models would require further experimentation to determine the minimal set of strong features that can predict performance accurately.

Love all, trust few, do wrong to none.

William Shakespeare

CHAPTER 9

CONCLUSIONS

In this thesis, we have examined heuristic search algorithms for combinatorial optimization problems and diversity maximization in the decision space. In the beginning of the thesis, we introduce combinatorial optimization and heuristic search. In Chapter 2, two well-known problems, which are travelling salesman problem and minimum vertex cover problem, are described in details with some population-based algorithms. The later on research focuses on theoretical and practical analysis of EAs for some trivial problems designed for theoretical investigation and these two combinatorial problems. A brief introduction of local search is included in Chapter 2 and followed with some fundamental knowledge about diversity in EAs. The theoretical investigation and some useful analysis tools are included as Chapter 3. These two chapters set the basis for the following studies.

The research into heuristic algorithms for the MVC problem detailed in Chapter 4 shows how the well-known fixed-parameter branching algorithms for the MVC problem can be turned into randomized initialization strategies which guarantee the probabilities of obtaining good initial solutions and meet our theoretical proof. Furthermore, we incorporate different branching rules into local search algorithms and present experimental results which show good results on some benchmark sets. Later we propose a new approach for scaling up local search algorithms for huge instances. The research is conducted on the MVC problem with the theoretical assumption that massive graphs are composed of different substructures which are not hard to be solved separately. Our approach is based on the parallel kernelization and reduces the instance size by multiple initial runs. This parallel kernelization approach presented in Chapter 4 is expected to be able to apply to other combinatorial

optimization problems with good local search solvers, e. g. Maximum Clique problem and Maximum Independent Set problem.

As discussed in Chapter 6 and 7, the population in EAs can be used to reserve a diverse set of solutions in which all solutions satisfy certain quality requirements. In Chapter 6, we conduct a first runtime analysis on the proposed $(\mu + 1)$ -EA_D which maximizes the population diversity after making sure all solutions in it have fulfilled certain quality criteria. How to design a proper population diversity measurement for different problems is discussed and followed with investigation into the computational complexity of $(\mu + 1)$ -EA_D on two classical benchmark problems OneMax and LeadingOnes. Based on these results, our study is moved on to a more general problem which is the MVC problem and we investigate some simple graph classes as examples. The investigations in this chapter set the basis for the analysis of diversity maximization for combinatorial optimization problems. Following the idea in this chapter, a runtime analysis on multi-objective problem is included in Chapter 7. We conduct a rigorous runtime analysis on the baseline algorithm $(\mu + 1)$ -SIBEA_D for the problem OneMinMax. The investigation can be furthered to other multi-objective optimization problems since the algorithm $(\mu + 1)$ -SIBEA_D is a general EA which can be applied to many other multi-objective problems as $(\mu + 1)$ -EA_D.

After the theoretical analysis of the diversity maximization in evolutionary algorithm, we apply the idea of optimizing diversity and guaranteeing the solution quality at the same time to a more practical problem. In Chapter 8, we introduce a new methodology of evolving easy/hard instances which are diverse in the feature space. In recent years, the investigation of heuristic search algorithms regarding the feature of certain problems has become very popular. The proposed algorithm follows the idea of $(\mu + 1)$ -EA_D and generates a set of problem instances which have achieved certain quality requirements. The population diversity measurement designed in the chapter provides good evaluation of the diversity over single or multiple feature values where feature value is usually a single numeric value. We conduct a case study of the travelling salesman problem and examine the behaviour of 2-OPT algorithm based on different TSP instances. The experimental results show that the large set of diverse instances generated by our approach can be classified into easy and hard instances when appropriate combination of features is considered in the experiments, which provides some guidance for instance hardness predication as the next step. Our approach can be generalized to other optimization problems with some appropriate features defined. The further research direction can focus on determining the minimal set of strong features which makes it possible to predict the performance of a certain algorithm accurately.

In conclusion, our research in parameterized analysis of heuristic search methods for combinatorial optimization problems introduces new branching rules for bounded search tree algorithms and provides new insights into solving huge problem instances with an existing solver for small instances. With rigorous runtime analyses of the decision-space diversity maximization in both single- and multi-objective optimization, we contribute to the theoretical understanding of diversity mechanisms in evolutionary optimization. Understanding the behaviour of heuristic search methods is a long time challenge to researchers and

practitioners. With the theoretical analysis of diversity optimization in solving optimization problems using evolutionary algorithms, we propose an approach of evolving hard/easy instances with diverse feature values for a certain algorithm. Moreover, the study focusing on the feature-based analysis of evolved hard/easy instances for a certain algorithm provides hints on how to build performance prediction models which leads to automatic algorithm selection and configuration.

BIBLIOGRAPHY

- [1] Emile Aarts and Jan K. Lenstra, eds. *Local Search in Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley, Chichester, UK, 1997.
- [2] Richa Agarwala, David L Applegate, Donna Maglott, Gregory D Schuler, and Alejandro A Schäffer. “A fast and scalable radiation hybrid map construction and integration strategy”. In: *Genome Research* 10.3 (2000), pp. 350–364.
- [3] R.D. Angel, W.L. Caudle, R. Noonan, and A. Whinston. “Computer-Assisted School Bus Scheduling”. In: *Management Science* 18.6 (1972), B-279–B-288.
- [4] David Applegate, William Cook, Sanjeeb Dash, and André Rohe. “Solution of a Min-Max Vehicle Routing Problem”. In: *INFORMS Journal on Computing* 14.2 (Apr. 2002), pp. 132–143.
- [5] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.
- [6] Sanjeev Arora. “Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems”. In: *J. ACM* 45.5 (Sept. 1998), pp. 753–782.
- [7] Sanjeev Arora. “Polynomial time approximation schemes for Euclidean TSP and other geometric problems”. In: *In Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS’96)*. 1996, pp. 2–11.
- [8] Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing Co., Inc., 2011.
- [9] David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. “Encyclopedia of Social Network Analysis and Mining”. In: New York, NY: Springer New York, 2014. Chap. Benchmarking for Graph Clustering and Partitioning, pp. 73–82.

- [10] Nikhil Bansal and Subhash Khot. "Inapproximability of Hypergraph Vertex Cover and Applications to Scheduling Problems". In: *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*. 2010, pp. 250–261.
- [11] Richard Bellman. "Dynamic Programming Treatment of the Travelling Salesman Problem". In: *J. ACM* 9.1 (Jan. 1962), pp. 61–63.
- [12] N. Beume, C. M. Fonseca, M. Lopez-Ibanez, L. Paquete, and J. Vahrenhold. "On the Complexity of Computing the Hypervolume Indicator". In: *IEEE Transactions on Evolutionary Computation* 13.5 (2009), pp. 1075–1082.
- [13] Christian Blum and Andrea Roli. "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Computing Surveys* 35.3 (Sept. 2003), pp. 268–308.
- [14] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. "Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem". In: *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*. 2010, pp. 1–10.
- [15] Shaowei Cai, Jinkun Lin, and Kaile Su. "Two Weighting Local Search for Minimum Vertex Cover". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. Austin, Texas, USA., 2015, pp. 1107–1113.
- [16] Shaowei Cai, Kaile Su, and Abdul Sattar. "Two New Local Search Strategies for Minimum Vertex Cover". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Toronto, Ontario, Canada., 2012.
- [17] Nachol Chaiyaratana, Theera Piroonratana, and Nuntapon Sangkawelert. "Effects of diversity control in single-objective and multi-objective genetic algorithms". In: *J. Heuristics* 13.1 (2007), pp. 1–34.
- [18] Jianer Chen, Iyad A. Kanj, and Weijia Jia. "Vertex Cover: Further Observations and Further Improvements". In: *Journal of Algorithms* 41.2 (2001), pp. 280–301.
- [19] Jianer Chen, Iyad A. Kanj, and Ge Xia. "Improved Parameterized Upper Bounds for Vertex Cover". In: *Mathematical Foundations of Computer Science 2006: 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006. Proceedings*. Ed. by Rastislav Kráľovič and Paweł Urzyczyn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 238–249.
- [20] Jianer Chen, Iyad A. Kanj, and Ge Xia. "Improved Upper Bounds for Vertex Cover". In: *Theoretical Computer Science* 411.40-42 (Sept. 2010), pp. 3736–3756.
- [21] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. DTIC Document, 1976.
- [22] Timothy W. McLain Christopher A. Bailey and Randal W. Beard. "Fuel saving strategies for separated spacecraft interferometry". In: *AIAA Guidance, Navigation and Control Conference*. 2000.
- [23] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

- [24] C. A. Coello Coello, G. Toscano Pulido, and E. Mezura Montes. "Current and Future Research Trends in Evolutionary Multiobjective Optimization". In: *Information Processing with Evolutionary Algorithms: From Industrial Applications to Academic Speculations*. Ed. by Xindong Wu, Lakhmi Jain, Manuel Graña, Richard J. Duro, Alicia d'Anjou, and Paul P. Wang. London: Springer London, 2005, pp. 213–231.
- [25] A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian. "Heuristics from nature for hard combinatorial optimization problems". In: *International Transactions in Operational Research* 3.1 (1996), pp. 1–21.
- [26] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. New York, NY, USA: John Wiley & Sons, Inc., 1998.
- [27] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297.
- [28] Dogan Corus, Per Kristian Lehre, Frank Neumann, and Mojgan Pourhassan. "A Parameterised Complexity Analysis of Bi-level Optimisation with Evolutionary Algorithms". In: *Evolutionary Computation* 24.1 (2015), pp. 183–203.
- [29] C. Cotta and P. Moscato. "A mixed evolutionary-statistical analysis of an algorithm's complexity". In: *Applied Mathematics Letters* 16.1 (2003), pp. 41–47.
- [30] G. A. Croes. "A Method for Solving Traveling-Salesman Problems". In: *Operations Research* 6.6 (1958), pp. 791–812.
- [31] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- [32] R. Dantzig G. and Johnson S. "Solution of a Large-Scale Traveling-Salesman Problem". In: *Journal of the Operations Research Society of America* 2.4 (1954), pp. 393–410.
- [33] Kenneth Alan De Jong. "An Analysis of the Behavior of a Class of Genetic Adaptive Systems." AAI7609381. PhD thesis. Ann Arbor, MI, USA, 1975.
- [34] Kalyanmoy Deb and Santosh Tiwari. "Omni-optimizer: A Procedure for Single and Multi-objective Optimization". In: *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*. EMO'05. Guanajuato, Mexico: Springer-Verlag, 2005, pp. 47–61.
- [35] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *Transactions Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197.
- [36] Irit Dinur and Samuel Safra. "On the Hardness of Approximating Minimum Vertex Cover". In: *Annals of Mathematics* 162 (2004), p. 2005.
- [37] Benjamin Doerr, Wanru Gao, and Frank Neumann. "Runtime Analysis of Evolutionary Diversity Maximization for OneMinMax". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*. 2016, pp. 557–564.

- [38] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. "Multiplicative Drift Analysis". In: *Algorithmica* 64.4 (2012), pp. 673–697.
- [39] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004.
- [40] Rod G. Downey and Michael R. Fellows. "Fixed-Parameter Tractability and Completeness I: Basic Results". In: *SIAM J. Comput.* 24.4 (Aug. 1995), pp. 873–921.
- [41] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer Publishing Company, Incorporated, 2013.
- [42] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the analysis of the (1+1) evolutionary algorithm". In: *Theoretical Computer Science* 276.1-2 (2002), pp. 51–81.
- [43] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.
- [44] Matthias Englert, Heiko Röglin, and Berthold Vöcking. "Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP". In: *Algorithmica* 68.1 (2014), pp. 190–264.
- [45] William Feller. *An Introduction to Probability Theory and Its Applications*. Vol. 1. Wiley, 1968.
- [46] C.-N. Fiechter. "A parallel tabu search algorithm for large traveling salesman problems". In: *Discrete Applied Mathematics* 51.3 (1994), pp. 243 –267.
- [47] Lawrence J Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, 1966.
- [48] Stephanie Forrest and Melanie Mitchell. "What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation". In: *Machine Learning* 13.2 (1993), pp. 285–319.
- [49] Tobias Friedrich, Nils Hebbinghaus, and Frank Neumann. "Comparison of simple diversity mechanisms on plateau functions". In: *Theoretical Computer Science* 410.26 (2009), pp. 2455–2462.
- [50] Tobias Friedrich, Pietro Simone Oliveto, Dirk Sudholt, and Carsten Witt. "Analysis of Diversity-Preserving Mechanisms for Global Exploration". In: *Evolutionary Computation* 17.4 (2009), pp. 455–476.
- [51] Wanru Gao, Tobias Friedrich, and Frank Neumann. "Fixed-Parameter Single Objective Search Heuristics for Minimum Vertex Cover". In: *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*. 2016, pp. 740–750.
- [52] Wanru Gao, Samadhi Nallaperuma, and Frank Neumann. "Feature-Based Diversity Optimization for Problem Instance Classification". In: *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*. 2016, pp. 869–879.
- [53] Wanru Gao and Frank Neumann. "Runtime analysis for maximizing population diversity in single-objective optimization". In: *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*. 2014, pp. 777–784.

- [54] Wanru Gao, Mojgan Pourhassan, and Frank Neumann. "Runtime Analysis of Evolutionary Diversity Optimization and the Vertex Cover Problem". In: *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*. 2015, pp. 1395–1396.
- [55] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
- [56] Oliver Giel and Ingo Wegener. "Evolutionary Algorithms and the Maximum Matching Problem". In: *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*. STACS '03. London, UK, UK: Springer-Verlag, 2003, pp. 415–426.
- [57] Fred Glover and Manuel Laguna. *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.
- [58] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [59] David E. Goldberg and Kalyanmoy Deb. "A comparative analysis of selection schemes used in genetic algorithms". In: *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 69–93.
- [60] David E. Goldberg and Jon Richardson. "Genetic Algorithms with Sharing for Multimodal Function Optimization". In: *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. Cambridge, Massachusetts, USA: L. Erlbaum Associates Inc., 1987, pp. 41–49.
- [61] Carla P. Gomes and Bart Selman. "Algorithm Portfolio Design: Theory vs. Practice". In: *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. UAI'97. Providence, Rhode Island: Morgan Kaufmann Publishers Inc., 1997, pp. 190–197.
- [62] Fernando C. Gomes, Cláudio N. Meneses, Panos M. Pardalos, and Gerardo Valdisio R. Viana. "Experimental Analysis of Approximation Algorithms for the Vertex Cover and Set Covering Problems". In: *Computers and Operations Research* 33.12 (2006). Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems, pp. 3520 – 3534.
- [63] M. Grötschel, M. Jünger, and G. Reinelt. "Optimal control of plotting and drilling machines: A case study". In: *Zeitschrift für Operations Research* 35.1 (1991), pp. 61–84.
- [64] Steve R. Gunn. *Support Vector Machines for Classification and Regression*. 1998.
- [65] Gregory Gutin, Anders Yeo, and Alexey Zverovich. "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the {TSP}". In: *Discrete Applied Mathematics* 117.1–3 (2002), pp. 81 –86.
- [66] Bruce Hajek. "Hitting-Time and Occupation-Time Bounds Implied by Drift Analysis with Applications". In: *Advances in Applied Probability* 14.3 (1982), pp. 502–525.
- [67] Eran Halperin. "Improved Approximation Algorithms for the Vertex Cover Problem in Graphs and Hypergraphs". In: *SIAM J. Comput.* 31.5 (May 2002), pp. 1608–1623.
- [68] Jun He and Xin Yao. "A Study of Drift Analysis for Estimating Computation Time of Evolutionary Algorithms". In: 3.1 (Mar. 2004), pp. 21–35.

- [69] Jun He and Xin Yao. "Erratum to: Drift analysis and average time complexity of evolutionary algorithms". In: *Artificial Intelligence* 140.1 (2002), pp. 245–248.
- [70] Jun He and Xin Yao. "From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms". In: *IEEE Transactions on Evolutionary Computation* 6.5 (2002), pp. 495–511.
- [71] Jun He and Xin Yao. "Towards an Analytic Framework for Analysing the Computation Time of Evolutionary Algorithms". In: *Artif. Intell.* 145.1-2 (Apr. 2003), pp. 59–97.
- [72] Michael Held and Richard M. Karp. "A Dynamic Programming Approach to Sequencing Problems". In: *Proceedings of the 1961 16th ACM National Meeting*. ACM '61. New York, NY, USA: ACM, 1961, pp. 71.201–71.204.
- [73] Keld Helsgaun. "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic". In: *European Journal of Operational Research* 126 (2000), pp. 106–130.
- [74] Jano I. van Hemert. "Evolving Combinatorial Problem Instances That Are Difficult to Solve". In: *Evol. Comput.* 14.4 (Dec. 2006), pp. 433–462.
- [75] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.
- [76] John H. Holland. *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1975.
- [77] J. N. Hooker. "Testing heuristics: We have it all wrong". In: *Journal of Heuristics* 1.1 (1995), pp. 33–42.
- [78] Holger H. Hoos and Thomas Stützle. *Stochastic local search : foundations and applications*. San Francisco, CA: Elsevier, 2005.
- [79] Christian Horoba and Frank Neumann. "Benefits and drawbacks for the use of epsilon-dominance in evolutionary multi-objective optimization". In: *Genetic and Evolutionary Computation Conference, GECCO , Proceedings, Atlanta, GA, USA, July 12-16, 2008*. 2008, pp. 641–648.
- [80] Thomas Jansen. *Analyzing Evolutionary Algorithms - The Computer Science Perspective*. Natural Computing Series. Springer, 2013, pp. 1–236.
- [81] Daniel Johannsen. "Random combinatorial structures and randomized search heuristics". eng. PhD thesis. Universität des Saarlandes, 2010.
- [82] Dura W. Sweeney John D. C. Little Katta G. Murty and Caroline Karel. "An Algorithm for the Traveling Salesman Problem". In: *Operations Research* 11.6 (1963), pp. 972–989.
- [83] David J. Johnson and Michael A. Trick, eds. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. Boston, MA, USA: American Mathematical Society, 1996.
- [84] David S Johnson and Lyle A McGeoch. "The traveling salesman problem: A case study in local optimization". In: (1997), pp. 215–310.

- [85] Terry Jones and Stephanie Forrest. "Fitness Distance Correlation As a Measure of Problem Difficulty for Genetic Algorithms". In: *Proceedings of the 6th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 184–192.
- [86] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. "Chapter 4 The traveling salesman problem". In: *Network Models*. Vol. 7. Handbooks in Operations Research and Management Science. Elsevier, 1995, pp. 225 –330.
- [87] Jorge Kanda, Andre Carvalho, Eduardo Hruschka, and Carlos Soares. "Selection of Algorithms to Solve Traveling Salesman Problems Using Meta-learning". In: *International Journal of Hybrid Intelligent Systems* 8.3 (Aug. 2011), pp. 117–128.
- [88] George Karakostas. "A Better Approximation Ratio for the Vertex Cover Problem". In: *ACM Transactions on Algorithms* 5.4 (Nov. 2009), 41:1–41:8.
- [89] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103.
- [90] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680.
- [91] Timo Kötzing, Frank Neumann, Heiko Röglin, and Carsten Witt. "Theoretical analysis of two ACO approaches for the traveling salesman problem". In: *Swarm Intelligence* 6.1 (2012), pp. 1–21.
- [92] C Koulamas, SR Antony, and R Jaen. "A survey of simulated annealing applications to operations research problems". In: *Omega* 22.1 (1994), pp. 41 –56.
- [93] John Koza. "Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm". In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 37–44.
- [94] Stefan Kratsch and Frank Neumann. "Fixed-Parameter Evolutionary Algorithms and the Vertex Cover Problem". In: *Algorithmica* 65.4 (2013), pp. 754–771.
- [95] Stefan Kratsch, Per Kristian Lehre, Frank Neumann, and Pietro Simone Oliveto. "Fixed Parameter Evolutionary Algorithms and Maximum Leaf Spanning Trees: A Matter of Mutation". In: *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*. 2010, pp. 204–213.
- [96] Gilbert Laporte. "The traveling salesman problem: An overview of exact and approximate algorithms". In: *European Journal of Operational Research* 59.2 (1992), pp. 231 –247.
- [97] Per Kristian Lehre. "Negative Drift in Populations". In: *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings Part I*. 2010, pp. 244–253.

- [98] Deming Lei. "Multi-objective production scheduling: a survey". In: *The International Journal of Advanced Manufacturing Technology* 43.9 (2008), pp. 926–938.
- [99] Yee Leung, Yong Gao, and Zongben Xu. "Degree of population diversity - a perspective on premature convergence in genetic algorithms and its Markov chain analysis". In: *IEEE Transactions on Neural Networks* 8.5 (1997), pp. 1165–1176.
- [100] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. "A Portfolio Approach to Algorithm Select". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence. IJCAI'03*. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 1542–1543.
- [101] S. Lin and B. W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". In: *Operations Research* 21.2 (Apr. 1973), pp. 498–516.
- [102] Marius Thomas Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. "AutoFolio: An Automatically Configured Algorithm Selector". In: *J. Artif. Intell. Res. (JAIR)* 53 (2015), pp. 745–778.
- [103] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. "Iterated Local Search". In: *Handbook of Metaheuristics*. Ed. by Fred Glover and Gary A. Kochenberger. Boston, MA: Springer US, 2003, pp. 320–353.
- [104] Jörg Lässig and Dirk Sudholt. "Analysis of speedups in parallel evolutionary algorithms and {EAs} for combinatorial optimization". In: *Theoretical Computer Science* 551 (2014), pp. 66–83.
- [105] Mirosław Malek, Mohan Guruswamy, Mihir Pandya, and Howard Owens. "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem". In: *Annals of Operations Research* 21.1 (1989), pp. 59–84.
- [106] Kim Fung Man, Kit Sang Tang, and Sam Kwong. "Genetic algorithms: concepts and applications [in engineering design]". In: *IEEE Transactions on Industrial Electronics* 43.5 (1996), pp. 519–534.
- [107] R.T. Marler and J.S. Arora. "Survey of multi-objective optimization methods for engineering". In: *Structural and Multidisciplinary Optimization* 26.6 (2004), pp. 369–395.
- [108] Kurt Mehlhorn. *Graph Algorithms and NP-completeness*. New York, NY, USA: Springer-Verlag New York, Inc., 1984.
- [109] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, Jakob Bossek, and Frank Neumann. "A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem". In: *Annals of Mathematics and Artificial Intelligence* 69.2 (2013), pp. 151–182.
- [110] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-7. 2015.
- [111] Zbigniew Michalewicz, Robert Hinterding, and Maciej Michalewicz. "Evolutionary algorithms". In: *Fuzzy evolutionary computation*. Springer, 1997, pp. 3–31.
- [112] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. New York, NY, USA: Cambridge University Press, 1995.

- [113] Heinz Mühlenbein. "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*. 1992, pp. 15–26.
- [114] Samadhi Nallaperuma, Andrew M. Sutton, and Frank Neumann. "Fixed-parameter evolutionary algorithms for the Euclidean Traveling Salesperson problem". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*. 2013, pp. 2037–2044.
- [115] Samadhi Nallaperuma, Markus Wagner, and Frank Neumann. "Analyzing the Effects of Instance Features and Algorithm Parameters for Max-Min Ant System and the Traveling Salesperson Problem". In: *Frontiers Robotics and AI 2015* (2015).
- [116] Samadhi Nallaperuma, Markus Wagner, and Frank Neumann. "Parameter Prediction Based on Features of Evolved Instances for Ant Colony Optimization and the Traveling Salesperson Problem". In: *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings*. 2014, pp. 100–109.
- [117] Samadhi Nallaperuma, Markus Wagner, Frank Neumann, Bernd Bischl, Olaf Mersmann, and Heike Trautmann. "A feature-based comparison of local search and the christofides algorithm for the travelling salesperson problem". In: *Foundations of Genetic Algorithms XII, FOGA '13, Adelaide, SA, Australia, January 16-20, 2013*. 2013, pp. 147–160.
- [118] Frank Neumann and Ingo Wegener. "Randomized Local Search, Evolutionary Algorithms, and the Minimum Spanning Tree Problem". In: *Theoretical Computer Science* 378.1 (June 2007), pp. 32–40.
- [119] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010.
- [120] Anh Quang Nguyen, Andrew M. Sutton, and Frank Neumann. "Population size matters: Rigorous runtime results for maximizing the hypervolume indicator". In: *Theoretical Computer Science* 561, Part A (2015). Genetic and Evolutionary Computation, pp. 24–36.
- [121] Pietro S. Oliveto, Jun He, and Xin Yao. "Analysis of the $(1 + 1)$ -EA for Finding Approximate Solutions to Vertex Cover Problems". In: *Transactions on Evolutionary Computation* 13.5 (Oct. 2009), pp. 1006–1029.
- [122] Pietro S. Oliveto, Jun He, and Xin Yao. "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results". In: *International Journal of Automation and Computing* 4.3 (2007), pp. 281–293.
- [123] Pietro S. Oliveto and Carsten Witt. "Simplified Drift Analysis for Proving Lower Bounds in Evolutionary Computation". In: *Algorithmica* 59.3 (2011), pp. 369–386.
- [124] Ibrahim H. Osman and Gilbert Laporte. "Metaheuristics: A bibliography". In: *Annals of Operations Research* 63.5 (1996), pp. 511–623.
- [125] Miliotis P. "Integer programming approaches to the travelling salesman problem". In: *Mathematical Programming* 1 (1976), pp. 367–378.

- [126] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.
- [127] Vangelis Th. Paschos. *Iste: Applications of Combinatorial Optimization*(1). Wiley-ISTE, 2013.
- [128] Wayne Pullan. “Phased local search for the maximum clique problem”. In: *Journal of Combinatorial Optimization* 12.3 (2006), pp. 303–323.
- [129] Wayne Pullan and Holger H. Hoos. “Dynamic Local Search for the Maximum Clique Problem”. In: *Journal of Artificial Intelligence Research* 25.1 (Feb. 2006), pp. 159–185.
- [130] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015.
- [131] Ingo Rechenberg. *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.
- [132] Colin R. Reeves. “Landscapes, operators and heuristic search”. In: *Annals of Operations Research* 86.0 (1999), pp. 473–490.
- [133] Gerhard Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Berlin, Heidelberg: Springer-Verlag, 1994.
- [134] John R. Rice. “The Algorithm Selection Problem”. In: *Advances in Computers* 15 (1976), pp. 65–118.
- [135] Silvia Richter, Malte Helmert, and Charles Gretton. “A Stochastic Local Search Approach to Vertex Cover”. English. In: *KI 2007: Advances in Artificial Intelligence*. Ed. by Joachim Hertzberg, Michael Beetz, and Roman Englert. Vol. 4667. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 412–426.
- [136] Ryan Rossi and Nesreen Ahmed. “The Network Data Repository with Interactive Graph Analytics and Visualization”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 2015, pp. 4292–4293.
- [137] Franz Rothlauf and David E. Goldberg. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, 2002.
- [138] Günter Rudolph. “Finite Markov Chain Results in Evolutionary Computation: A Tour D’Horizon”. In: *Fundamenta Informaticae* 35.1-4 (Aug. 1998), pp. 67–89.
- [139] Günter Rudolph, Boris Naujoks, and Mike Preuss. “Capabilities of EMOA to Detect and Preserve Equivalent Pareto Subsets”. In: *Proceedings of the 4th International Conference on Evolutionary Multi-criterion Optimization*. EMO’07. Matsushima, Japan: Springer-Verlag, 2007, pp. 36–50.
- [140] Maytham Safar and Sami Habib. “Hard Constrained Vertex-Cover Communication Algorithm for WSN”. In: *Embedded and Ubiquitous Computing, International Conference, EUC 2007, Taipei, Taiwan, December 17-20, 2007, Proceedings*. 2007, pp. 635–649.
- [141] Hussain Aziz Saleh and Rachid Chelouah. “The design of the global navigation satellite system surveying networks using genetic algorithms”. In: *Engineering Applications of Artificial Intelligence* 17.1 (2004), pp. 111 –122.

- [142] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications". In: *Data Mining and Knowledge Discovery* 2.2 (June 1998), pp. 169–194.
- [143] J. David Schaffer. "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms". In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 93–100.
- [144] Tommaso Schiavinotto and Thomas Stützle. "A Review of Metrics on Permutations for Search Landscape Analysis". In: *Computers & Operations Research* 34.10 (Oct. 2007), pp. 3143–3153.
- [145] Ofer M. Shir, Mike Preuss, Boris Naujoks, and Michael T. M. Emmerich. "Enhancing Decision Space Diversity in Evolutionary Multiobjective Algorithms". In: *Evolutionary Multi-Criterion Optimization, 5th International Conference, EMO 2009, Nantes, France, April 7-10, 2009. Proceedings*. 2009, pp. 95–109.
- [146] Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. "Understanding TSP difficulty by learning from evolved instances". In: *4th International Conference on Learning and Intelligent Optimization (LION)*. LION'10. Springer, 2010, pp. 266–280.
- [147] Kate Smith-Miles and Leo Lopes. "Measuring Instance Difficulty for Combinatorial Optimization Problems". In: *Computers and Operations Research* 39.5 (May 2012), pp. 875–889.
- [148] Kate Smith-Miles and Thomas T. Tan. "Measuring algorithm footprints in instance space". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*. 2012, pp. 1–8.
- [149] N. Srinivas and Kalyanmoy Deb. "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms". In: *Evolutionary Computation* 2.3 (Sept. 1994), pp. 221–248.
- [150] Andrew M. Sutton, Frank Neumann, and Samadhi Nallaperuma. "Parameterized Runtime Analyses of Evolutionary Algorithms for the Planar Euclidean Traveling Salesperson Problem". In: *Evolutionary Computation* 22.4 (2014), pp. 595–628.
- [151] Ma. Guadalupe Castillo Tapia and Carlos A. Coello Coello. "Applications of multi-objective evolutionary algorithms in economics and finance: A survey". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore*. 2007, pp. 532–539.
- [152] Andrea Toffolo and Ernesto Benini. "Genetic Diversity As an Objective in Multi-objective Evolutionary Algorithms". In: *Evolutionary Computation* 11.2 (May 2003), pp. 151–167.
- [153] Raymond Tran, Junhua Wu, Christopher Denison, Thomas Ackling, Markus Wagner, and Frank Neumann. "Fast and effective multi-objective optimisation of wind turbine placement". In: *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*. 2013, pp. 1381–1388.
- [154] Tamara Ulrich, Johannes Bader, and Lothar Thiele. "Defining and Optimizing Indicator-Based Diversity Measures in Multiobjective Search". In: *Parallel Problem Solving from*

- Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*. 2010, pp. 707–717.
- [155] Tamara Ulrich, Johannes Bader, and Eckart Zitzler. “Integrating decision space diversity into hypervolume-based multiobjective search”. In: *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*. 2010, pp. 455–462.
- [156] Tamara Ulrich and Lothar Thiele. “Maximizing population diversity in single-objective optimization”. In: *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*. 2011, pp. 641–648.
- [157] Rasmus K. Ursem. “Diversity-Guided Evolutionary Algorithms”. In: *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain, September 7-11, 2002, Proceedings*. 2002, pp. 462–474.
- [158] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.
- [159] Ingo Wegener. “Methods for the Analysis of Evolutionary Algorithms on Pseudo-Boolean Functions”. In: *Evolutionary Optimization*. Boston, MA: Springer US, 2002, pp. 349–369.
- [160] Ingo Wegener and Carsten Witt. “On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions”. In: *Journal of Discrete Algorithms* 3.1 (2005), pp. 61–78.
- [161] Paul J. Schweitzer William T. McCormick Jr. and Thomas W. White. “Problem Decomposition and Data Reorganization by a Clustering Technique”. In: *Operations Research* 20.5 (1972), pp. 993–1009.
- [162] Carsten Witt. “Runtime Analysis of the $(\mu + 1)$ EA on Simple Pseudo-Boolean Functions”. In: *Evolutionary Computation* 14.1 (2006), pp. 65–86.
- [163] David H. Wolpert and William G. Macready. “No Free Lunch Theorems for Optimization”. In: *Transactions on Evolutionary Computation* 1.1 (Apr. 1997), pp. 67–82.
- [164] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. “A Simple Model to Generate Hard Satisfiable Instances”. In: *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*. Edinburgh, Scotland, UK, 2005, pp. 337–342.
- [165] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “SATzilla: portfolio-based algorithm selection for SAT”. In: *Journal of Artificial Intelligence Research* 32.1 (June 2008), pp. 565–606.
- [166] Qingfu Zhang and Hui Li. “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”. In: *IEEE Transactions on Evolutionary Computation* 11.6 (2007), pp. 712–731.
- [167] Feifei Zheng, Angus Simpson, and Aaron Zecchin. “Improving the Efficiency of Multi-objective Evolutionary Algorithms Through Decomposition”. In: *Environmental Modelling and Software* 69.C (July 2015), pp. 240–252.
- [168] Eckart Zitzler and Simon Künzli. “Indicator-Based Selection in Multiobjective Search”. In: *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*. 2004, pp. 832–842.

-
- [169] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. "A Tutorial on Evolutionary Multiobjective Optimization". In: *Metaheuristics for Multiobjective Optimisation*. Ed. by Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 3–37.
- [170] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization". In: *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*. Ed. by K.C. Giannakoglou et al. International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.